2020

Identifying and Disentangling Interleaved Activities of Daily Living from Sensor Data

Eoin Rogers
Technological University Dublin, eoin.rogers@tudublin.ie

Identifying and Disentangling Interleaved Activities of Daily Living from Sensor Data

by

Eoin Rogers

Thesis submitted for the degree of

Doctor of Philosophy

SCHOOL OF COMPUTER SCIENCE

Technological University Dublin

Supervisors: Dr. Robert J. Ross
Prof. John D. Kelleher

October 2020

Declaration

I certify that this thesis which I now submit for examination for the award of Doctor of Philosophy, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work.

This thesis was prepared according to the regulations for postgraduate study by research of the Technological University Dublin and has not been submitted in whole or in part for an award in any other Institute or University.

The work reported on in this thesis conforms to the principles and requirements of the TU Dublin's guidelines for ethics in research.

TU Dublin has permission to keep, to lend or to copy this thesis in whole or in part, on condition that any such use of the material of the thesis be duly acknowledged.

Signature: _____ Date: _____

Abstract

Activity discovery (AD) refers to the unsupervised extraction of structured activity data from a stream of sensor readings in a real-world or virtual environment. Activity discovery is part of the broader topic of *activity recognition*, which has potential uses in fields as varied as social work and elder care, psychology and intrusion detection. Since activity recognition datasets are both hard to come by, and very time consuming to label, the development of reliable activity discovery systems could be of significant utility to the researchers and developers working in the field, as well as to the wider machine learning community.

This thesis focuses on the investigation of activity discovery systems that can deal with *interleaving*, which refers to the phenomenon of continuous switching between multiple high-level activities over a short period of time. This is a common characteristic of the real-world datastreams that activity discovery systems have to deal with, but it is one that is unfortunately often left unaddressed in the existing literature.

As part of the research presented in this thesis, the fact that activities exist at multiple levels of abstraction is highlighted. A single activity is often a constituent element of a larger, more complex activity, and in turn has constituents of its own that are activities. Thus this investiga-

tion necessarily considers activity discovery systems that can find these hierarchies.

The primary contribution of this thesis is the development and evaluation of an activity discovery system that is capable of identifying interleaved activities in sequential data. Starting from a baseline system implemented using a topic model, novel approaches are proposed making use of modern *language models* taken from the field of natural language processing, before moving on to more advanced language modelling that can handle complex, interleaved data. As well as the identification of activities, the thesis also proposes the *abstraction* of activities into larger, more complex activities. This allows for the construction of hierarchies of activities that more closely reflect the complex inherent structure of activities present in real-world datasets compared to other approaches.

The thesis also discusses a number of important issues relating to the evaluation of activity discovery systems, and examines how existing evaluation metrics may at times be misleading. This includes highlighting the existence of differing abstraction issues in activity discovery evaluation, and suggestions for how this problem can be mitigated. Finally, alternative evaluation metrics are investigated.

Naturally, this dissertation does not fully solve the problem of activity discovery, and work remains to be done. However, a number of the most pressing issues that affect real-world activity discovery systems are tackled head-on, and show that useful progress can indeed be made

on them. This work aims to benefit systems that are as “clean slate" as possible, and hence incorporate no domain-specific knowledge. This is perhaps somewhat of an artificial handicap to impose in this problem domain, but it does have the advantage of making this work applicable to as broad a range of domains as possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer systems are becoming increasingly ubiquitous in domestic environments, as is the use of home automation technologies (Khedekar et al., 2017). As the technologies involved become more sophisticated and expensive, one can reasonably expect their capabilities to also improve. Within the domain, one possible technology that could become promising in the near future is *activity recognition* (Ravi et al., 2005; Wang et al., 2019): the practice of determining the activities currently being carried out by one or more agents in a real-world environment via algorithmic means. This could potentially be useful for tasks such as data-gathering, elder care, crowd control, anomaly detection and a range of similar tasks, assuming that the potentially major issues relating to privacy and trust issues can be overcome (Xu et al., 2019; Liu et al., 2019; Wang et al., 2020; Mohan et al., 2019).

Modern activity recognition is highly sophisticated, and often utilises state-of-the-art machine learning techniques (Cook et al., 2013; Viard

et al., 2020). Since these models tend to require supervised training of some sort, this leads to the age-old problem of having to obtain good quality, labelled datasets. In order to simplify and speed up the process of gathering such datasets, and also to fix a number of issues known to affect the labelling of such, some researchers have developed the field of *activity discovery* (Cook et al., 2013).

An activity discovery system can in effect be viewed as an unsupervised activity recognition system, and thus it takes as input a dataset consisting purely of raw sensor readings (i.e. one which is unlabelled) and outputs a plausible labelling for the activities observed. Incorporating semantic information provided by the user into the discovered activities is certainly possible (for example, an activity that triggers a flurry of sensor readings near the kettle in a kitchen might be reasonably labelled something like *Make-Coffee* or *Make-Tea*). In spite of this, most existing activity discovery research tends to focus on the simple identification of the activities (i.e. determining that an activity relating to the kettle takes place after dinner each evening) (Gjoreski and Roggen, 2017; Nguyen et al., 2017; Chen et al., 2016), since this is the heart of any activity discovery system that would be deployed in the real world.

Activity discovery is an ongoing area of research (Safavi et al., 2020; Gjoreski and Roggen, 2017), and this thesis specifically focuses on examining the use of activity discovery to deal with datasets and scenarios that are highly *interleaved* – meaning datasets in which the agent(s) in

the environment in question are carrying out multiple activities at a time (or more accurately, switching between activities rapidly, in a manner similar to how operating systems context switch between processes).

This introduction will first detail the above information, before moving on to outlining the remainder of the thesis and proposing a novel approach to this problem.

1.1 Activities

Before going any further, it would make sense to have a working definition of an activity. Given an agent in a particular environment, one can define an *activity* as being a sequence of low-level *actions* that the agent carries out with a degree of regularity. One might object to this on the grounds that this definition provides no indication of *how* regular a sequence of actions have to be in order to qualify as an activity, but this is at least somewhat application specific. An action sequence may only occur a handful of times to qualify as an activity for tasks like anomaly detection (since the activities that are of interest in such a case are, by definition, rare), but they may have to occur nearly daily to qualify as an *activity of daily living* (ADL) – a term widely used to describe the activities recognised by systems deployed in homes and offices and originating from medical measures of patient independence (Katz et al., 1970). Some examples of activities that could be expected to appear in a

dataset gathered in a domestic environment could be *Preparing-Dinner*, *Preparing-Breakfast*, *Showering*, *Using-Toilet* etc.

Figure 1.1, which is taken from Roggen et al. (2010), gives an example of the sort of set-up used to produce the datasets used in this field. A kitchen area has been kitted out with a suite of sensors. For example, there is a sensor (labeled *A6_7* in the dataset, as indicated in Figure 1.1) attached to the door, which activates whenever the door opens. Some of the sensors produce discrete (often binary) outputs, while others produce complex continuous outputs. For instance, many small objects in the kitchen, such as items of cutlery, have accelerometers attached to them, which allows for the recording of the cutlery's motion and the derivation of its orientation in space. A person, who is labelled as the *subject* near the top left corner of Figure 1.1, can be seen carrying out activities in the environment. As they do so, two experimenters, one near the top right of the image and another near the centre bottom, observe them, and annotate the activities that they carry out (a process called *online annotation*). Thus, for machine learning purposes, the sensor readings become features and the annotations become labels in the final dataset. The dotted white line visible on the diagram shows the path that the subject took through the environment.

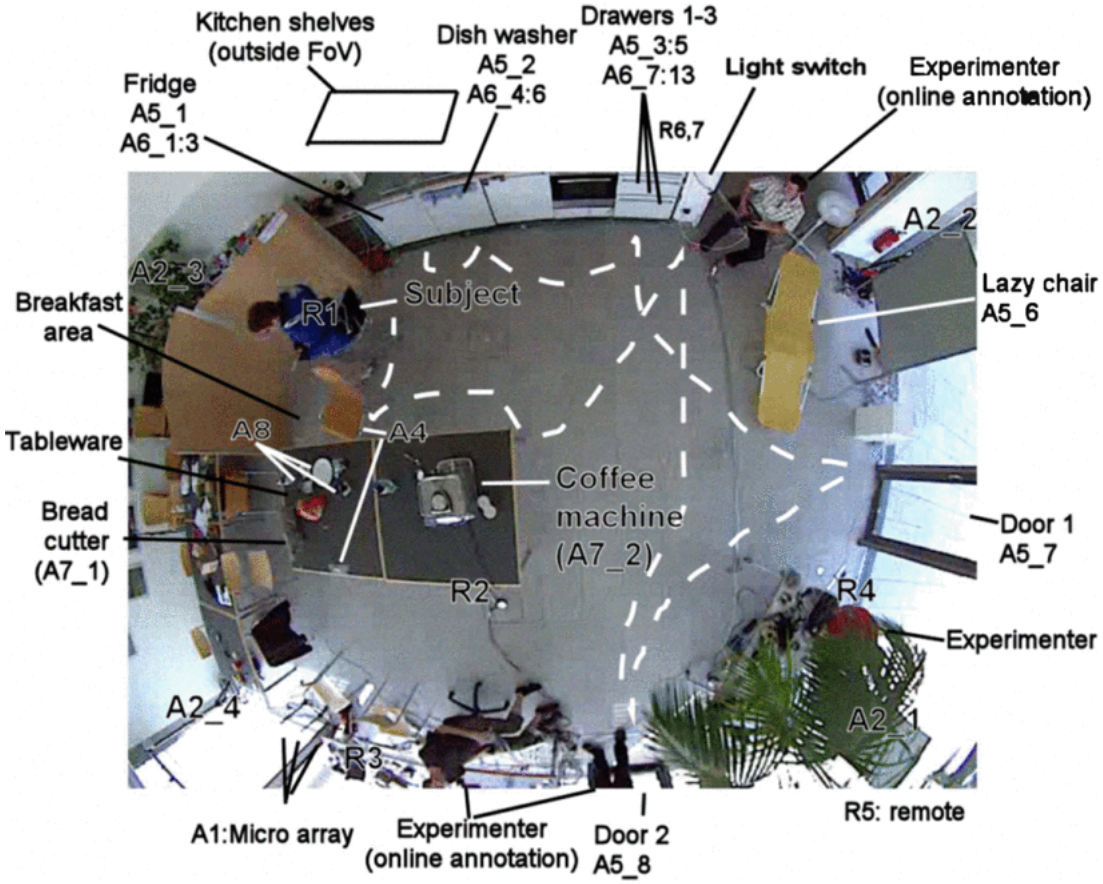


Figure 1.1
Gathering a dataset for activity recognition and/or discovery. Taken from Roggen et al. (2010)

1.2 Activity recognition

Given a dataset constructed in the manner illustrated in Figure 1.1, activity recognition can be defined as the development and deployment of machine learning models that are able to map the sensor readings in the environment to the activity labels that the experimenter's annotations provide, and the field of research surrounding such systems (Ravi et al., 2005). A brief overview of this field will now be presented.

Van Kasteren et al. (2008) is an excellent example of a prototypical activity recognition system. Interestingly, the paper presents an end-to-

end system – the entire activity recognition pipeline, from the setup of the sensors to gather data, through developing the recognition model itself, to evaluation are all covered in the paper. The dataset consisted of readings collected from 14 binary-valued sensors installed in a three bedroom apartment for a period of 28 days. This data is then pre-processed by splitting the dataset into time slices of a fixed length Δt, where the i^{th} time slice is converted into a binary vector $\mathbf{x}_i$. The j^{th} sensor has a corresponding binary value in the vector x_{ij}. Once collected such labelled data was preprocessed and used to train a supervised classifier based on models such as the Hidden Markov Model.

After the preprocessing stage, the resulting data was used to train two models which were compared to against each other. The first was a *hidden Markov model* (HMM), where the $\mathbf{x}_i$ vectors are taken as visible states and the activity as the hidden state to be inferred. The inference was carried out after training using the Viterbi algorithm (Viterbi, 1967). The second proposed model was a *conditional random field* (CRF), specifically a *linear-chain CRF*, with potential functions defined on the current and previous hidden state ($\psi(y_{i-1}, y_i)$) and the current hidden and visible states ($\psi(y_i, \mathbf{x}_i)$), resulting in a structure similar to an HMM.

In the years since the publication of the Van Kasteren et al. (2008) paper, the increasing ubiquity of on-body and wearable devices such as smartwatches and smartphones has provided an attractive means of gathering detailed datasets from users with minimal intrusion. Kwapisz

et al. (2011) describes an activity recognition system that splits smartphone accelerometer data into 10 second slices and computes statistical summaries of the real-valued sensor data found in each slice (averages, standard deviation, etc.). These are then used as inputs to a selection of traditional machine learning systems, with the authors trying decision trees, neural networks and logistic regression. Of these, the neural network (specifically a multilayer perceptron) was the most accurate, with an average accuracy of 91.7%.

One interesting problem resulting from the use of technologies like smartphones is that users may not carry the device in a way that is optimal for gathering data useful in an activity recognition context. For example, a phone might not be carried around on a user's person when at home, or it may be placed in a handbag or rucksack when going out. In order to help deal with this problem, Sun et al. (2010) propose building a system which can determine the orientation and position of the phone using the accelerometer data, and which then selects a model pre-trained for the orientation in question.

Weiss et al. (2016) describe activity recognition using smartwatches instead of phones, arguing that watches are better positioned to capture accurate information about a user's movements. The results in the paper seem to back up this argument. The system they propose is otherwise very similar to the Kwapisz et al. (2011) system described above.

One aspect of activity recognition that is often overlooked is how best

to present the data to users. An evaluation of graphical formats to present information about Activities of Daily Living (ADLs) to caregivers is presented by Byrne et al. (2020). The authors employed the *System Usability Scale*, a widely used standard for evaluating the usability of a software system by means of user surveys. The results indicate that user's prefer short-term activity data to be represented in the form of a pie chart, and longer term data to be shown in the form of a bar chart.

Li et al. (2018) addresses the problem of activity recognition in the context of real-time, online systems. Such systems need to be able to correctly classify an activity shortly after the activity begins, rather than by processing data offline after the event has taken place. The approach presented, named *COBRA* maintains a sliding window over activities that is always maintained at a fixed temporal period. As sensor events occur and are passed to the system, older events are dropped from the window to maintain this period. The bulk of the activity classification in the system is done using a logistic regression model, but *instantaneous activities* (those with a short temporal length), which may not correspond to a large number of sensor events, are instead detected using a hand-engineered algorithm.

Recent progress in the field of deep learning has also found its way into the activity recognition literature. A wide-ranging survey of this is presented by Wang et al. (2019), covering both datasets and activity recognition systems. Some highlights from the discussed systems in-

clude *DeepEar*, which recognises activities using audio data (Lane et al., 2015), and a system to determine the severity of Parkinson's disease by recognising behaviours symptomatic of the disease as activities (Hammerla et al., 2015). More recently, Thapa et al. (2020) has proposed a sophisticated bidirectional LSTM-based model for activity recognition in interleaved situations.

Finally, Wu et al. (2019) present a very complex deep learning-based system for recognising activities in groups of agents. The system takes raw video data as input, and extracts bounding boxes over individual people visible in it using the *Inception v3* deep learning model (Szegedy et al., 2016). Regions of interest are identified in the bounding boxes, and these are used to construct an *actor relation graph* (ARG), depicting the relations and interactions between individual people in the video. A specialised type of network called a *graph convolutional network* (GCN) is used to process the graphs and perform reasoning over the interactions in them. The output from this is used as input to an ordinary feedforward classifier to recognise the activities. This entire model can be trained end-to-end by backpropagation – the system takes video data as input, and outputs activities, learning the internal graph representations during the training stage itself.

1.3 Activity discovery

This thesis focuses not on activity recognition in general, but on a subfield of activity recognition called *activity discovery*. Activity discovery refers to the *unsupervised* detection of activities from raw sensor data. That is, an activity discovery system attempts to identify activities without having any prior knowledge about what activities may be present in a dataset.

The value of activity discovery as a field stems from the fact that there are a number of issues with human annotation in the field of activity recognition, which will be explicated in detail in the coming chapters. For now, however, it can be observed that Roggen et al. (2010) have elected to use two experimenters to annotate their data, which is likely to be an attempt to minimise the risk of issues cropping up with the annotation. Thus, without even having gone far in investigating activity recognition, one already see that accurate annotation can be a problem. Mitigating this by annotating the data automatically may therefore be a worthwhile goal to pursue.

An idealised activity discovery system would take an input consisting purely of an unlabelled dataset (i.e. the sensor readings from the dataset and nothing more), and return a full sensible annotation, similar to the one that the experimenters would have produced. In reality, the task is rescoped by relaxing the requirement of semantic information that would

be required for a full automated annotation. Thus, an activity discovery system is primarily concerned with identifying sequences of actions that occur with a degree a regularity, and need not concern itself with their meaning. For example, in a kitchen environment, one can imagine that meals are prepared one or more times a day. This would be reflected in the sensor readings, where sensors relating to cupboards, cutlery, the fridge and perhaps oven or microwave occur reliably at around the same time each day. Identifying this repeating pattern of sensor events is the essence of activity discovery.

In theory, it should indeed be possible to add semantic information to this output, in other words to identify that this activity observed daily is indeed some sort of food preparation. Doing this requires incorporating knowledge about the real-world into such a system. Such a system would thus have to have a repertoire of possible activities that it is likely to encounter in use, and a list of the *entities* that are likely to be associated with each of them. For example, a possible activity like *Using-Toilet* could be associated with the bathroom door, taps in the bathroom, any light switch(es) in the bathroom and of course the toilet itself. When sensors are installed in the environment, each sensor is then associated with an entity. So a toilet flush sensor could be associated with the toilet, a sensor detecting that a door is open that is attached to the bathroom door would be associated with the bathroom door and so forth. Although such a system is possible, this thesis will instead focus solely on the

sequence identification task outlined previously, without any sort of semantic labelling.

There are a number of other assumptions that will be made to narrow the scope of this thesis. It is assumed that the sensor streams that will be provided as input to the activity discovery models that will be considered are simple sensors, which produce a stream of low-level, discrete *events*, as opposed to more complex sensor modalities such as cameras. This has the advantage of alleviating privacy concerns and reducing the quantity of pre-processing that is required. It also means that the systems discussed are quite general, and can work with many different sensor types, which will hopefully make them applicable to a wide range of uses and scenarios. The systems proposed will also not make use of temporal information, which is present in many datasets. This also improves privacy, but it also makes the problem of activity discovery noticeably harder.

To reduce the complexity of the problem, there will be a number of simplifying assumptions made about the activity discovery task that the models will be performing. This thesis mostly ignores the problem of multi-agent activity discovery. Although a particular focus is given to the discovery of activities that occur in parallel (as will be discussed in Section 1.3.1, which would allow activities being carried out by multiple agents to be identified and tracked, the models proposed have no way of knowing the number of agents in the environment being monitored.

Finally, the proposed systems will also be assumed to be mostly offline rather than online. This means that they will not be expected to discover activities in real-time – rather it will be assumed that the input to the system is an entire dataset that has been gathered prior to the activity discovery system being given the data. This keeps the scope of the problem manageable, since any one of these requirements could themselves be a major topic of investigation. In many ways, it is assumed that the systems proposed in this thesis would not be a *replacement* for activity recognition (as some of the literature in the field seems to assume), but rather could be used to provided automated annotation to datasets of human activity, which could then be used as a ground truth to train more traditional activity recognition systems.

1.3.1 Interleaving

One of the most persistent issues that affect real-world activity discovery systems is *interleaving*. The worldview outlined above assumes a sort of linear progression of tasks, where one activity is carried out to completion, and another activity will not start until then. However, this is highly unrealistic: in reality people may only partially complete one task before moving on to another, with the intention of returning to the first task at a later time. Thus, activities are interleaved between each other, and this presents a major challenge for activity discovery in the real world (Kim et al., 2009; Cook et al., 2013).

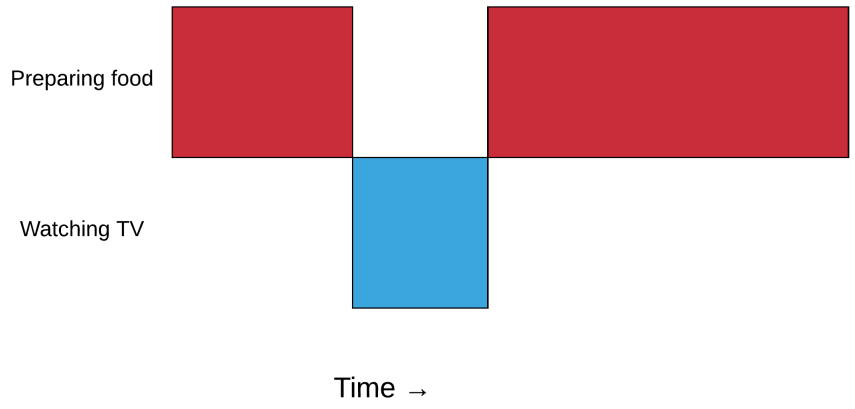


Figure 1.2
Interleaving occurs when a subject suspends one activity temporarily with the intention to return to it later. While the food cooks, the subject here watches TV until the cooking is finished. They then resume the food preparation activity.

An example of this is illustrated in Figure 1.2. The example shows only the annotations for an imagined dataset. One can see that a subject begins an activity called *Preparing food*. They do this for a period of time, before something prompts them to suspend the task. For example, it might be the case that they have placed the food into the oven. Whatever the reason, they decide to carry out a *Watching TV* activity for a time. After a number of minutes, they then return to the food preparation task. It is not possible see what happens after this in Figure 1.2, but it is possible, for example, that the subject would go back to watching TV with their meal after preparing it. Interleaving is a difficult problem because it introduces the issue of *long-range dependencies* (Angeli et al., 2015; Dieng et al., 2016; Mahalunkar and Kelleher, 2018b, 2019). Resolving this problem remains a major challenge for activity discovery, and a major focus of this thesis.

1.4 Research questions

This thesis is an investigation of the use of activity discovery on interleaved datasets. As noted above, activity discovery is distinguished from traditional activity recognition by being unsupervised. Interleaving continues to be a significant challenge for activity discovery systems, and thus continues to be a major focus of research in the field (Raeiszadeh et al., 2019). Unfortunately, in spite of the difficulties that interleaving presents for activity discovery systems in general, it is one which is ripe for further investigation, and one for which progress seems to be quite slow. Towards that end, this thesis specifically addresses the following research questions:

1. **Is it possible to construct activity discovery systems that perform well on interleaved data?** Most activity discovery systems operating on real world data can reasonably expect that data to contain interleaved activities on a regular basis. Thus, dealing with interleaving is very important if the output from such systems is to be trusted. Unfortunately, interleaved data is often a point of weakness of existing AD systems (and, indeed, of activity recognition more generally). This thesis aims to investigate how activity discovery can be done in a way which is robust to interleaving.

2. **Can anything be learned from existing approaches to pattern analysis to construct more robust activity discovery systems?**

A wealth of existing literature dedicated to pattern analysis already exists in fields such as natural language processing. Unfortunately, there seems to be little cross-fertilisation of these ideas into activity discovery. The experiments presented in the thesis will determine if such approaches could be beneficial to activity discovery. Such systems should show a clear improvement over existing systems in the field if they are to be accepted.

3. **How can activity discovery systems be evaluated in a fair manner?** As will also become clear later on in this thesis, the *evaluation* of activity discovery systems remains a major stumbling block to progress in the field. Evaluation of such systems is inherently difficult, for a number of reasons that will be outlined in future chapters (see Chapter 7 in particular). The thesis contain an investigation into more robust and fair evaluation metrics.

4. **How can hierarchies of activities be constructed?** As will be observed in later chapters, the *hierarchical nature of activities* is a major issue that is often not addressed by the existing literature. Most real-world activities can be understood as consisting of smaller constituent activities. For example, the act of *making dinner* might consist of *chopping vegetables*. Constructing models and systems that take this into account explicitly might help make the output of the system more meaningful, and also helps address the

evaluation issues mentioned above.

1.5 Contributions of this thesis

Driven by the above research questions, the work summarised in this dissertation has tackled the activity discovery problem from a number of different perspectives. This work has resulted in the following contributions that are summarised by this thesis:

- An investigation into the use of modern techniques from the field of natural language processing (NLP) and related fields to discover activities in a novel manner, which allows for the explicit disentaglement of interleaved activities. The proposed approach indirectly models the interleaving observed in the data, allowing it to deal with interleaved data in a more principled manner than many existing systems can (Chapters 4, 5 and 6).

- An identification of a number of issues with existing “standard" approaches to evaluating activity discovery systems. To help rectify these issues, the wider use of metrics from natural language processing (NLP) is proposed, as well as the further incorporation of minimum description length (MDL) principles for AD evaluation (Chapter 7 and to a lesser extent, Chapter 2).

- A model for the construction of hierarchical trees of activities, where low-level activities are discovered within sequences of events,

and higher-level activities are composed of a combination of events and lower-level activities. This both helps reduce the impact of interleaving, and will prove useful for evaluation (Chapter 3 onwards).

It is hoped that these contributions assist in the development and evaluation of activity discovery systems that are robust to noise, and are able to deal with the interleaving problem in a principled manner without sacrificing performance.

1.6 Publications

The following publications have been made from the research conducted for this thesis:

- **Using topic modelling algorithms for hierarchical activity discovery** (2016) Eoin Rogers, Robert Ross, John D. Kelleher. *Ambient Intelligence- Software and Applications – 7th International Symposium on Ambient Intelligence (ISAmI 2016)*, pages 41–48
- **Towards a Deep Learning-based Activity Discovery System** (2016) Eoin Rogers, Robert Ross, John D. Kelleher. *Proceedings of the 24th Irish Conference on Artificial Intelligence and Cognitive Science (AICS 2016)*, pages 184–191
- **Tackling the Interleaving Problem in Activity Discovery** (2017)

Eoin Rogers, Robert Ross, John D. Kelleher. *Trends in Cyber-Physical Multi-Agent Systems. The PAAMS Collection - 15th International Conference, PAAMS 2017*, pages 313–314

- **Evaluating Sequence Discovery Systems in an Abstraction-Aware Manner** (2018) Eoin Rogers, Robert Ross, John D. Kelleher. *Artificial Intelligence Applications and Innovations (14th IFIP WG 12.5 International Conference, AIAI 2018)*, pages 261–272

- **Language Model Co-occurrence Linking for Interleaved Activity Discovery** (2020) Eoin Rogers, Robert Ross, John D. Kelleher. *Machine Learning for Networking (Second IFIP TC 6 International Conference, MLN 2019)*, pages 70–84

- **Modelling Interleaved Activities using Language Models** (2020) Eoin Rogers, Robert Ross, John D. Kelleher. *Proceedings of the 34th International ECMS Conference on Modelling and Simulation, ECMS 2020*, pages 183–189

1.7 Organisation of this thesis

The layout of the remainder of this thesis is as follows. Chapter 2 provides some background information about activity discovery, and discusses some important prior work in the area.

Chapter 3 presents the first activity discovery system to be docu-

mented in this thesis, based on a generative topic model. Although an interesting system in its own right, the major novelty in terms of modeling proposed by this thesis is the development of a generalization of the standard language modelling task that is appropriate for activity discovery with interleaved data. A sequence of refined implementations of this generalised language model based approach to activity discovery is presented in Chapters 4, 5 and 6, and so this topic model based system will primarily be used as a baseline when evaluating the more novel approaches proposed later in the thesis. The latter of these models significantly outperforms the original, and exhibits very high performance on interleaved datasets.

Chapter 7 will discuss the problem of evaluating activity discovery systems. This is an issue that appears to be very challenging, and surprisingly poorly addressed in the existing literature. Finally, Chapter 8 concludes the thesis with some final remarks and suggestions for future work.

Chapter 2

Finding Structure in Sequential Data

At a fundamental level, this thesis is about finding patterns in data. This is a very broad and complex task, but fortunately it possible to simplify the task into something tractable by looking at the characteristics of the data used and the information that is to be produced from it.

To begin with, the datasets that will be provided as input to an activity discovery system are *sequential*. This means that they consist of a contiguous sequence $D = \langle d_1, d_2, \ldots, d_n \rangle$ of *events* $d_i \in \Sigma$, each of which is drawn from a set of *event types* Σ. By contrast, it is expected that the output will have some richer structure than simply being a sequence of tokens, in that it will identify sub-sequences of the dataset that are instances of activities. This makes activity discovery a special case of the more general concept of *structured prediction* (Bakir et al., 2007; Smith, 2011), the use of machine learning in contexts in which the output is a complex data structure. This contrasts with the traditional focus on classification and regression which focuses on atomic single or multiple

target variables.

In general, activity discovery systems are expected to be unsupervised, in the sense that the user of the system provides *only* the dataset as input, and the model is expected to automatically extract the structure from this dataset. The user is not expected to give training labels, such as examples of what an activity is expected to look like. Thus, one can say that activity discovery is a form of *unsupervised structured prediction*. This chapter will expand on this definition. Starting in Section 2.1, a more detailed outline of structured prediction is provided, followed by an introduction to some common notation that will be used throughout the thesis. Following this will be a broad overview of various techniques used for structured prediction in the general field of sequential data, covering topic models (Section 2.2), motif discovery (Section 2.3), grammar induction (Section 2.4) and minimum description length (Section 2.5). In Section 2.6, the focus is narrowed to activity discovery specifically, detailing approaches to activity discovery found in the literature, before providing a formal definition of activity discovery in Section 2.7. Section 2.8 details the datasets that have been used in this research, followed by some commentary on the presented approaches in Section 2.9. The chapter concludes with a short summary (Section 2.10).

2.1 Introduction to sequence analysis

A concise explanation of how one can view sequential data as having an underlying structure is found in the opening chapter of Smith (2011). Smith focuses on structured prediction tasks on language, which is one form of sequential data. The notation used here follows Smith, which in turn follows standard notational conventions. Where possible, this notation through this entire thesis, although it will be re-introduced with minor variations throughout in order to better illuminate salient concepts.

Assume the existence of an alphabet, Σ, consisting of a finite set of characters or tokens that can be used to construct sentences. Σ^n denotes the set of all strings of length n that one can construct using this alphabet. For example, given an alphabet $\Sigma = \{a, b, c\}$, then the set of strings of length 2, $\Sigma^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$. Σ^* denotes the set of *all* possible strings. Regardless of the size of the alphabet, Σ^* will be an infinite set, since it is the union of all Σ^n sets for every positive integer n, which is an infinite number of sets. A *language* L is then defined in the most abstract sense as a subset of the set of all strings, i.e. $L \subseteq \Sigma^*$. Thus, each element of L is a string that is considered a valid "sentence" in the language. This is a very general definition of a language, and includes both natural human languages, as well as artificial and formal languages such as mathematical notation, programming and markup languages.

It is possible to carry out structured prediction tasks on sentences

from this language. The canonical example of this would be to produce a parse tree for a natural language sentence (see Figure 2.1). The tasks involved may be simpler, however. Some examples detailed by Smith (2011) are:

- **Sequential prediction**: Suppose $s \in L$ is a sentence in L. $s_{i:j}$ denotes the substring of s from index i to j. Given a partial string $s_{1:j}$, the task of trying to predict the next character, s_{j+1}, is called *sequential prediction*. Although seemingly trivial, this will turn out to be an essential component of the work presented in this thesis.

- **Sequence segmentation**: A sentence s of length $\#s$ can be broken up into one or more contiguous, non-overlapping *segments*. Trying to do this automatically has a number of uses in natural language processing. For example, segmenting the start and end of words (*word segmentation*) can be a formidable task for languages such as Chinese, since the written language will frequently not delimit words by means of a space (Li and Yuan, 1998). Another example could be tasks such as *named entity recognition*, where segments are used to encapsulate entities that are of interest to an information extraction pipeline, for example proper nouns or periods of time.

- **Sequence labeling**: Applying a label to the elements of a string is a common task. For example, *part-of-speech (POS)* tagging is frequently used in the NLP community (Ratnaparkhi, 1996; Plank

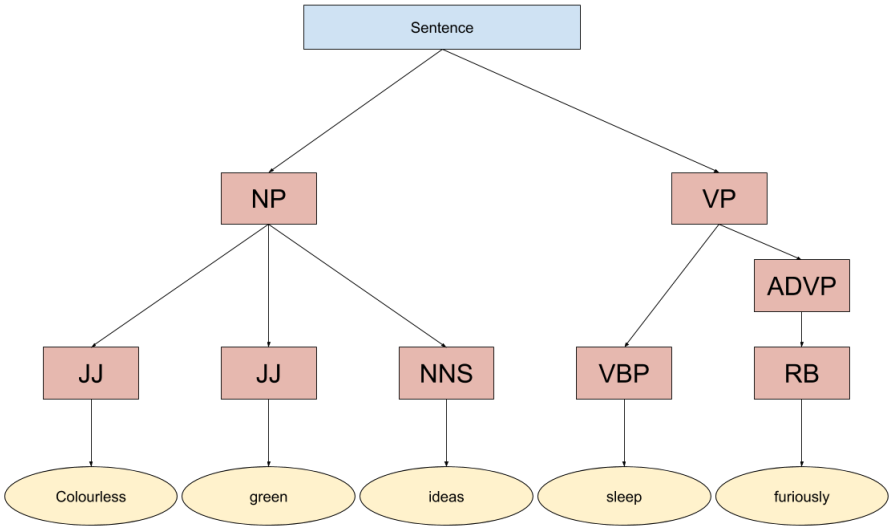


Figure 2.1
A parse tree for an English sentence.

et al., 2016). It is not unusual for this to be combined with other tasks. For example, word segmentation might be followed by POS tagging. This allows for the construction of very complex outputs.

As noted before, the focus of this thesis is on activity discovery rather than natural language processing. However, activity discovery can be viewed as being analogous to the sequence prediction tasks described above. Specifically, activity discovery is much like being given a set of natural language sentences, and then parsing them without any prior knowledge of the language. This is an immensely more difficult task than parsing based on a given grammar or dataset of sample parses. Here, the dataset is a large string s, drawn from an alphabet Σ, where each $\sigma \in \Sigma$ is a *sensor event*, which are the most primitive inputs to the activity recognition models this thesis will be considering.

2.2 Topic modeling for activity discovery

One commonly used approach to activity recognition, which generally performs well, is *topic modeling*. Given a dataset D consisting of n discrete *documents*, topic modeling is the problem of classifying these documents into one of k *topics*, where two documents will share the same topic if they are about a similar topic. This is much like how a news website might classify articles into categories like crime, politics, sport and so forth. The difference is that topic modeling is unsupervised, and the user typically does not pre-define topics. So, topic modeling systems are expected to take a dataset of uncategorised articles, and automatically group them into coherent categories. Of course, there is no expectation that a topic modelling system can attach semantic information to these categories beyond assigning documents to them. For example, it is not assumed that the topic modelling systems assign semantically appropriate names (such as the news website category names mentioned previously) to the document groupings it identifies.

Huynh et al. (2008) present a typical example of an activity discovery system, based on a form of topic modeling called *latent Dirichlet allocation* (LDA), which will be discussed in more detail in Chapter 3. A dataset was gathered from a subject wearing two sensors as they went about their daily lives. These sensors differ slightly from the setup that was outlined at the beginning of this chapter in that they are producing

data that is *continuous*, i.e. a time series of real numbers, rather than a sequence of simple sensor events drawn from an alphabet Σ. Like many topic modelling-based approaches to activity discovery, Huynh et al. (2008)'s model assumes that the input dataset is discrete. Initially, the authors of the paper resolved this problem simply by manually converting the data into annotated, low-level actions such as *talking on the phone* or *driving a car*. The raw dataset was split into segments, each containing all of the sensor events that took place over a fixed period of time. This temporal length was the same for all segments. Two experiments were presented by the authors. As a proof-of-concept, each segment was manually annotated with a ground truth, and the individual segments were fed into an LDA topic model as documents for classification. There was a strong correlation observed between the discovered topics and the activities in the labelled dataset.

However, this process still required access to a manually annotated dataset, which is clearly undesirable in a field where fully unsupervised approaches tend to be favoured (Vahdatpour et al., 2009). For this reason, the authors also proposed automatically converting the (continuous) dataset into discrete events by running *another* small temporal window over the raw continuous dataset, extracting various features from the sensor readings in these windows (with features mostly consisting of statistical summaries of the sensor readings, such as mean values, standard deviation and the like) to produce a 13-dimension vector for each window.

This vector was fed as input to a traditional clustering algorithm, with the resulting classifications being used as discrete event types (corresponding to Σ in the notation used in this chapter). This resulted in a discrete dataset that can be fed as input into the LDA model used in the previous experiment. The resulting system achieves an F-1 score of about 75% on their dataset.

More recently, Seiter et al. (2014) proposed a similar system. The authors of this paper put a lot of emphasis on having a sophisticated approach to discretisation. Similar to the work reported in Huynh et al. (2008), Seiter et al. collected a continuous dataset and then discretised it by segmenting it into fixed length time periods and annotating each time period with an event type. Seiter et al. defined 36 event types (which the authors call *activity primitives*). This discretisation is done using simple rules developed by the authors: for example, one rule is active if a sensor detects one of the user's arms accelerate at a rate exceeding a particular threshold over the course of one second. The definition, implementation, and testing of these rules requires a significant quantity of user input to work, and requires manual modification before it can be applied to other domains. The resulting discrete dataset is again fed into a topic model to discover activities contained within it. Seiter et al. report that their approach is quite robust when provided with noisy datasets as input. This makes it well adapted to activity recognition in situations where individuals are prone to changing their routine from day-to-day. On top

of that, this system introduces two further topic modelling algorithms. These will be covered further in Section 3.2, once the basic operation of LDA in has been discussed some detail.

2.3 Motif discovery

The term *motif* is used in a number of fields to denote a type of recurring pattern (Machanick and Bailey, 2011; Welch, 2013; Lonardi and Patel, 2002; Sporns and Kötter, 2004). Since an activity can also be construed as a type of recurring pattern, algorithms that already exist for the detection or discovery of motifs could be useful for activity discovery. One example of a motif discovery algorithm is the *MM algorithm* presented by Bailey et al. (1994), and used primarily in bioinformatics applications. This algorithm assumes that a dataset D is split up into n overlapping subsequences, each of length W. Each subsequence is assumed to be a either part of a single motif (which occurs multiple times in the dataset), or a background subsequence (i.e. not part of a motif). It is assumed that D is generated by a mixture model, with a mixing parameter $\lambda = (\lambda_1, \lambda_2)$, where λ_1 is the probability of a subsequence being generated as part of a motif, and $\lambda_2 = 1 - \lambda_1$ the probability of a subsequence being generated as part of the background. It is again assumed that the entire dataset consists of characters drawn from a know alphabet Σ, of length L. The data are generated by the unknown parameters $\theta = (\theta_1, \theta_2)$, where θ_1 is

the parameter used to generate motif subsequences, and θ_2 to generate background subsequences. θ_1 is a matrix of dimension $W \times L$, such that each $\theta_{1_{ij}}$ in the matrix is the probability that the i^{th} item in the motif is the jth character in the alphabet Σ. By contrast, θ_2 is simply a vector of length L, where each θ_{2_j} is the probability that the jth character in the alphabet will appear in any position of the background. The values of θ are determined using the well known expectation-maximisation algorithm (Dempster et al., 1977). Note that an important aspect of this approach is that it explicitly models the *order* of the events (alphabet characters) that make up the motif. The topic modeling approaches described earlier typically do not do this, and order could hypothetically be important to the discovery of activities (for instance, turning on a tap to wash one's hands is presumably more likely after flushing the toilet). The focus on finding a *single* recurring motif, however, could be a problem, since activity discovery usually involves finding *several* distinct motifs that exist together in the dataset.

Nonetheless, it is worth investigating motif discovery algorithms as they are an active area of research, and one in which interesting problems that affect sequence discovery as a whole have been investigated. One such interesting problem, and a proposed solution to it is highlighted by Buhler and Tompa (2002). The problem, called the *planted motif problem* or *(l, d)-motif problem*, originates from the field of bioinformatics, where motif discovery algorithms are used to find recurring sequences

in DNA, which could turn out to be biologically significant (Pevzner et al., 2000). Assume the existence of a dataset consisting of n DNA sequences, each of length L. In each sequence, a single instance of a motif y has been planted. The motif is of length l, and each instance has exactly d substitutions applied to the base form of the motif. The planted motif problem, then, is simply to identify this base form of the motif. Although conceptually simple, this turns out to be a problem for many existing sequence discovery algorithms (Rajasekaran et al., 2005). The reason for this is that the algorithms generally depend on splitting each of the n sequences into subsequences with a sliding window of length l, and then measuring the similarity between windows from separate sequences. Large sets of similar sequences are then analysed using some form of clustering algorithm to produce the motif. The similarity is typically determined using the *Hamming distance* measure (Hamming, 1950), where the Hamming distance $H(a, b)$ between two strings a and b of a shared length $|a|$ is defined as in equation 2.1.

$$H(a,b) = \sum_{i=1...|a|} \left\{ \begin{array}{l} 1 \text{ if } a_i \neq b_i \\ 0 \text{ otherwise} \end{array} \right\} \tag{2.1}$$

Suppose that one investigated a particular instance of this problem where $l = 4$ and $d = 2$ over the alphabet of DNA base chemicals $\Sigma = \{A, T, C, G\}$. If the base form of the motif (which is what the motif discovery system should recover) is $ATTA$, two possible instances of

the motif that one could expect to observe in the dataset are $CGTA$ and $ATCG$. Observe, however, the value of the Hamming distance between these two strings: $H(CGTA, ATCG) = 4$, which is twice the value of d. For any possible size of d, the maximum Hamming distance between any two instances of the motif could be as high as $2d$. Thus, two sequences a and b could be instances of the same motif for all distances $H(a, b) \leq 2d$. This can prove too computationally expensive even for relatively small values of l and d. Pevzner et al. (2000) found the problem to be intractable for values as low as $l = 15$ and $d = 4$.

Buhler and Tompa (2002) propose to get around this problem using a form of location-sensitive hashing they call *projection*. Each observed sequence is placed into a bin computed by the hashing algorithm. This algorithm is iterative, and runs over x iterations. For each iteration, k numbers are drawn randomly (and without replacement) from $\{1, 2, \ldots, l\}$ to form the set K_x. For each subsequence in each sequence, the ${K_x}_i^{\text{th}}$ character is taken for each of the k values in K_x, and concatenated into a smaller string called a projection. The projection for iteration x of subsequence s will be denoted as $Proj_{xk}(s)$. If the number of times a particular projection is observed exceeds some constant, it is hypothesised that the projection is part of the motif. The concrete sequences that produced the projection are retrieved from the dataset, and the motif discovery proceeds using a clustering algorithm as before. Buhler and Tompa (2002) also proposed principled approaches for determining the

optimal values of k and x from the given values of l and d, as well as the size of the dataset.

A variant of Buhler and Tompa's approach is proposed in Chiu et al. (2003) for use on continuous data: i.e., rather than having a discrete alphabet Σ as in the formalisation above, each element of each sequence is a real number. This allows for the discovery of motifs in a wide range of complex data, for example biological data relating to things like a diabetic's blood sugar levels over time, or data drawn for a non-biological domain such as telemetric data from an aircraft or bathymetric data from a ship. To do this, the raw continuous waveform is transformed into a series of discrete *levels* using a discretisation algorithm called *Piecewise Aggregate Approximation* (PAA). The PAA output is shown in Figure 2.2 as a light grey line over the main original continuous data. The levels are then clustered together into symbols (represented in Figure 2.2 as bold letters and colours). Each sequence can then be represented as a string of these symbols. Buhler and Tompa's algorithm is then used to produce the final output.

An interesting variant of the motif discovery problem is the idea of a *contextual motif* presented in Fox et al. (2017). This is defined as a motif in which there is a separate set of contextual data, which could provide clues about where motifs are present in the dataset. For example, Fox et al. (2017) show that a motif discovery system running over continuous blood sugar level data (as described above) becomes substantially more

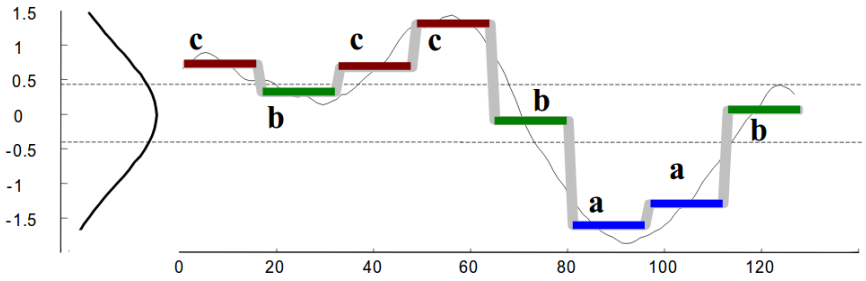


Figure 2.2
Converting continuous waveform data into a discrete sequence of symbols using PAA and symbol clustering. Taken from Chiu et al. (2003)

accurate when given contextual data about when the person the data was produced from eats. The authors of the paper present a simple algorithm for discovering contexts automatically from a dataset that does not come with contextual information on its own. This involves the following steps:

```
# Split the dataset using a sliding window
# of length w
dataset = split_dataset(input_dataset, w)
proto_contexts = []
for window in dataset:
    proto_context = context_discovery_algorithm(window)
    proto_contexts += [proto_context]
C = cluster(proto_contexts)
return C
```

The authors propose their own generative model for use as a context discovery algorithm, although they note that existing approaches such

as a hidden Markov model could be used also, where the emissions are treated as the continuous dataset and the hidden states as contexts, although they find their model works better. Like all generative models, this is presented as a probabilistic explanation of how the data observed in the dataset came to be. Each observed datum is assumed to be generated by selecting a context, followed by a motif given a context, followed by observed data. Although this is clearly not how datasets containing motifs come to be in reality, it provides a probabilistic framework in which motifs and contexts can be discovered together.

2.4 Grammar induction

Although motifs and topic modeling have both been used directly in the literature to carry out activity discovery, this section outlines an approach that *could potentially* be used for activity discovery, and which will turn out to be conceptually important in later chapters. *Grammar induction* refers to the automatic generation of formal grammars for a given language using only positive and negative examples, or perhaps even just positive examples, of the language in use (D'Ulizia et al., 2011). This is a rather challenging problem: in the 1960s, Gold (1967) found that, at least in the limiting case, not even a regular language can be induced from positive examples alone. In spite of this, this section will only concentrate on examining grammar induction approaches and algorithms

that require positive examples. It is hypothesised that these are most useful for the purposes of activity recognition, since activity discovery systems generally do not have access to explicit negative examples when training. The intention is that the nonterminal symbols in the grammar given as output will (at least in some cases) correspond to activities.

To begin with, a brief recap of the concept of a formal context-free grammar is provided in order to ensure that the notation used is clear. For readers unfamiliar with the concept, chapters 1 and 2 of Sipser (2012), or possibly Chomsky (1956), are recommended as introductions to the general concept. A *context-free grammar* G is defined as a 4-tuple of the form $G = (X, \Sigma, S, R)$. Σ is a set of *terminals*, i.e. things (letters or words, either level of abstraction will do) that can appear in a valid sentence produced by the grammar. By contrast, X is a set of *nonterminals*. A nonterminal is a name used to denote a sequence of terminals from Σ. S is the *starting nonterminal*, and it must be the case that $S \in X$. R is a set of *rewriting rules* or *productions* of the form $\alpha \mapsto \gamma$, where $\alpha \in X$ is called the *left-hand side* of the rule, and $\gamma \in (X \cup \Sigma)^*$ is called the *right-hand side* of the rule. The rule can be understood as saying that the nonterminal α denotes a sequence of terminals γ. Note that although γ can contain nonterminals also, these nonterminals will reduce to terminals by means of another rule, so defining nonterminals as names for a sequence of terminals still makes sense. It is presumed that a sentence is generated by starting at the starting nonterminal S and

deriving a sentence using Algorithm 1.

```
sentence = S
while nonterminals in sentence:
    α = the first nonterminal present
    γ = RHS of α
    sentence.replace(α, r)
```

Algorithm 1: Generating a sentence from the nonterminal S

Thus, grammar induction refers to techniques to try to induce such grammars from examples of the language, which are presumed to have been generated by the grammar. In spite of the results obtained by Gold (1967), it is possible to reasonably induce context-free grammars from examples in many cases. In an ideal scenario, a grammar induction algorithm would be able to derive even richer grammars than just context-free ones (recursively enumerable languages, for example, which can only be recognised by a Turing machine). In reality, most researchers in the field stick to context free grammars because there is no known way to derive recursively enumerable grammars in a reasonable amount of time.

A small selection of grammar induction algorithms will now be outlined to give the reader an idea of how they work. *Alignment-based Learning* (ABL) was proposed by Van Zaanen (2000). This is based on a simple intuition that is often used in grammar induction. Consider two sentences, sentence one $s_1 = I\ like\ grammar\ induction$ and sentence two $s_2 = I\ like\ long\ walks\ on\ the\ beach$. The first two words of these sentences are obviously the same, but the remaining are

not. Without knowing anything else about language, one could see that the phrase $I\ like$ can start a sentence, and can be followed by either $grammar\ induction$ or $long\ walks\ on\ the\ beach$. One can thus map these two phrases to a single nonterminal symbol via the two rules shown in Equations 2.2 and 2.3.

$$\alpha \mapsto grammar\ induction \tag{2.2}$$

$$\alpha \mapsto long\ walks\ on\ the\ beach \tag{2.3}$$

This allows both sentences to be represented as $I\ like\ \alpha$. This can thus be subsumed into a new nonterminal, which can then be used as a starting nonterminal, as in Equation 2.4.

$$S \mapsto I\ like\ \alpha \tag{2.4}$$

ABL works in two stages. The first stage, called *alignment learning*, involves finding the shared words between sentences as was done above. It also accounts, however, for the fact that there may be *slots* for words or phrases between words that do not become visible until later, and so it will have to revise the data structures in memory before committing to them. For instance, if a sentence is later found with the phrase $I\ really\ like$, the rule in Equation 2.4 will have to be rewritten to $S \mapsto I\ \beta\ like\ \alpha$, and two new rules will have to be added to R of

the form $\beta \mapsto really$ and $\beta \mapsto \emptyset$. The second stage, *selection learning*, involves actually distilling the rules in a way that resolves any conflicts in a principled manner.

Déjean (2000) has proposed the *Architecture for Learning Linguistic Structures* (ALLiS). This is based on a concept called *theory refinement*, where an initial grammar is developed using a simple algorithm, and subsequently extended to better fit the dataset. The initial grammar is generated by mapping the most common part-of-speech for each word to hypothesised nonterminals based on context. The grammar is then extended greedily, using hand-coded rules for generalising and specialising existing rules developed by the authors. This algorithm has an obvious disadvantage compared to the others, as it requires parts-of-speech to be tagged prior to the algorithm running. This is less useful for activity discovery, since it would require the development of some analogue of part-of-speech tags for sensor events. This is not necessarily impossible (perhaps parts-of-speech could correspond to location or other information about the sensor that generated the event, or perhaps something similar to the context extraction system proposed by Fox et al. (2017) and discussed in Section 2.3), but it does increase the complexity of the resulting algorithm.

Perhaps the gold standard in current approaches to grammar induction for the last number of years has been an algorithm called *Automatic Distillation of Structure* (ADiOS) (Horn et al., 2004; Solan et al., 2005).

ADiOS is based on three stages, the last two of which are repeated iteratively. The first step involves loading the entire corpus into a graph data structure, in which each word appears as a vertex in the graph (and only one vertex, regardless of how often the word occurs in the dataset), and an edge is inserted connecting vertex A to vertex B iff the word corresponding to B occurs after the word corresponding to A in the corpus. In other words, given the dataset above, one would expect the vertex representing the word “I" to be connected to the vertex representing the word “like".

The second stage, *pattern distillation*, uses a technique called *Motif Extraction* (MeX) which identifies common phrases in the graph in a context-sensitive manner. Given a particular path through the graph $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$, the idea is to search this path for significant patterns. To do this, Solan et al. (2005) defines a value called the *right-moving fan-in ratio*, R_R, which is defined in Equation 2.5, for each vertex V.

$$R_R(V) = \frac{Number\ of\ edges\ from\ Pre(V)\ to\ V}{Number\ of\ edges\ entering\ Pre(V)} \tag{2.5}$$

where $Pre(V)$ is the predecessor vertex of of V, so $Pre(C) = B$.

Likewise, *left-moving fan in ratio*, R_L, can be defined as Equation 2.6

$$R_L(V) = \frac{Number\ of\ edges\ from\ Succ(V)\ to\ V}{Number\ of\ edges\ entering\ Succ(V)} \tag{2.6}$$

where $Succ(V)$ is the successor vertex of of V, so $Succ(C) = D$.

ADiOS is derived from a somewhat unintuitive observation: given a subset of a path, say $B \rightarrow C \rightarrow D$, one can say this subset is a statistically significant pattern by computing the left- and right-moving fan-ins for each vertex in the path. If the left fan-in *peaks* at B, and the right fan-in peaks at D (see Figure 2.3), this means that this subset $B \rightarrow C \rightarrow D$ is thus a pattern discovered by the MeX process. Thus, a new vertex is added to the graph representing this pattern. The edges representing all sentences containing $B \rightarrow C \rightarrow D$ are then re-routed through the new node, and removed from the original $B \rightarrow C \rightarrow D$. Edges are inserted from the predecessors of B and the new node, and from the successors of D and the new node. Note that *the existing vertexes for the subpath are not removed*, since they might appear in other contexts outside of the pattern (if they do not, they will be orphaned vertexes anyway, so it does not matter what is done from a purely algorithmic viewpoint, although obviously removing them would be more computationally efficient with regards to space complexity).

The final stage is called *finding equivalence classes*. This involves identifying any class of vertexes such that each vertex shares *exactly* the same set of preceding and successor vertexes, meaning that they can be replaced with one another at will and the resulting sentence would still be valid. An example is shown in Figure 2.4. Here, the words $cat, dog, horse$ are found to belong to one equivalence class, and the

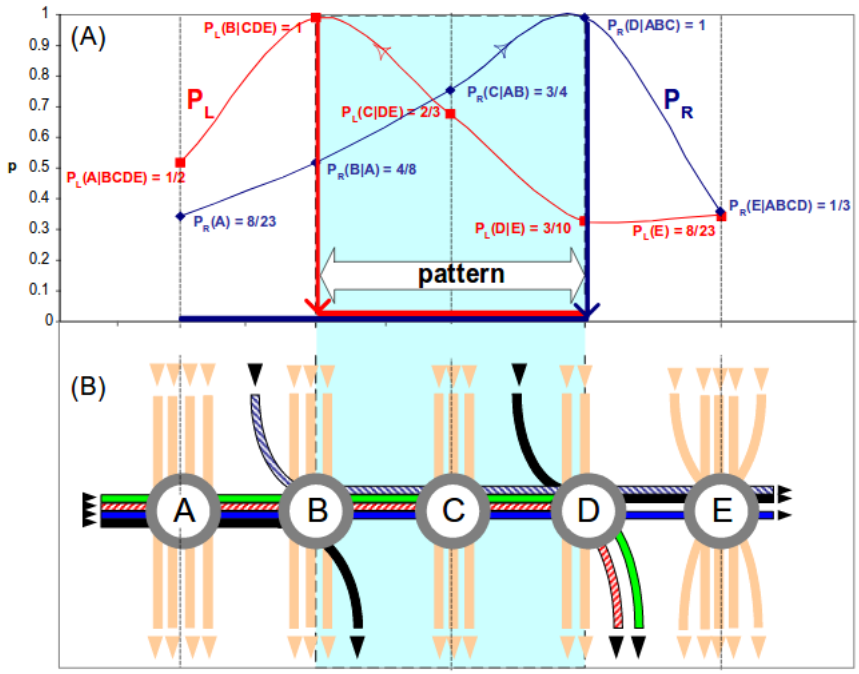


Figure 2.3
Identifying patterns based on the peaks of the right- and left-moving fan-in ratios. The left-moving fan-in ratio P_L peaks at vertex B, while the right-moving fan-in ratio P_R peaks at vertex D. This indicates that $B \rightarrow C \rightarrow D$ forms a pattern, which can be converted to a new node. Taken from Horn et al. (2004)

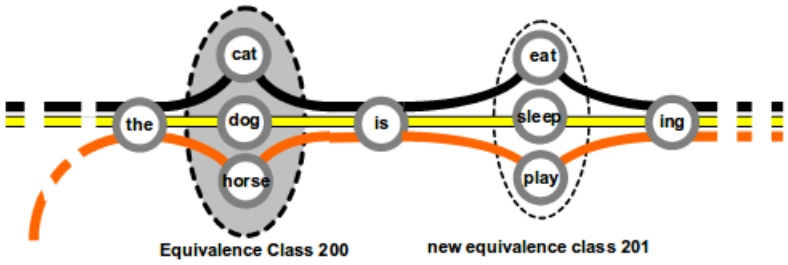


Figure 2.4
Identifying and replacing equivalence classes in a graph. Taken from Horn et al. (2004)

words $eat, sleep, play$ are found to belong to another. For each equivalence class found, the *entire* class should be replaced with an a new single vertex α, and a rule should be added to the output grammar of the form $\alpha \mapsto \gamma_i$ for each γ_i in the equivalence class.

Sequitur (Nevill-Manning and Witten, 1997) is an algorithm that can

rapidly build a context-free grammar from a given input text, and thus is arguably a form of grammar induction. What it does not do, however, that one would expect a full grammar induction algorithm to do, is to produce a *generalised* grammar; i.e., one that allows new, novel sentences to be generated. The grammars output by Sequitur can only be used to re-create the input data. Nonetheless, it will be briefly described because it is closely related to other work described in this section.

Sequitur accepts a single input (i.e., it has no prior knowledge of concepts like sentences, which contrasts with some of the algorithms outlined above), and converts this into an output grammar. It works by sliding a window w of length two (a *bigram*) across the input D. It keeps a copy of each substring/bigram w_i that it observes as it runs. If any substring recurs (i.e. matches a previously observed substring), a new nonterminal α is created, and the rule $\alpha \mapsto w_i$ is added to the ruleset R. Every instance of w_i in the original dataset is then replaced by the new nonterminal α.

This process is applied iteratively: once the sliding window has traversed the entire dataset, Sequitur starts the process over again, allowing it to build a tree-like hierarchical structure. This is repeated until no further sliding windows that appear at least twice in the dataset are observed.

On occasion, the hierarchical building process can end up accepting rules into the grammar that only appear once, in spite of the sliding

window only adding rules if they appear at least twice. For example, imagine the dataset is the string of symbols $abcabcb$. When Sequitur is first run, it observes the digrams shown in Equation 2.7.

$$ab, bc, ca, ab, bc, cb \tag{2.7}$$

The bigram ab has now been observed twice (w_1 and w_4), and thus Sequitur creates a new rule $\gamma \mapsto ab$ and changes the dataset to $\gamma c \gamma c b$. The second pass now takes place, resulting in the bigrams $\gamma c, c\gamma, \gamma c, cb$. Here, one can see that γc appears twice, so a new rule is created: $\delta \mapsto \gamma c$. The dataset is updated to $\delta \delta b$. At this point one can see that there is a problem: the first rule, $\gamma \mapsto ab$ is now only used once: inside the right-hand side of the δ rule. Sequitur resolves this problem by removing the γ nonterminal, and then replacing it with ab explicitly in the rule for the δ nonterminal, so the rule now looks like $\delta \mapsto abc$. Thus, Sequitur has a means of producing rules containing more than just two elements if their creation results in a smaller overall output.

Creating a smaller overall output is in effect the primary goal of the algorithm (i.e. Sequitur is a form of *lossless compression*). This might seem like a goal unrelated to the subject of this thesis, but in fact machine learning and compression have a deep mathematical link, as suggested by the minimum description length principle, which will now be discussed.

2.5 Minimum description length

One grammar induction algorithm that was intentionally left out of the discussion in Section 2.4 is called *eGrids* (Petasis et al., 2004). The reason for this is that understanding the full algorithm is difficult without a further piece of background knowledge, namely a basic understanding of the concept of *minimum description length* (MDL). It will thus be described MDL here, followed by a discussion of the eGrids algorithm.

MDL was introduced by Rissanen (1978) as a formalisation of Occam's razor. Occam's razor refers to the philosophical idea that, given a number of hypotheses that explain a set of observed data equally well, one should generally favour the *simplest* hypothesis. This is based on the interesting insight that learning (some would even argue intelligence in general) is mathematically equivalent to data compression. The entire point of machine learning is to build a model that generalises over the dataset and even over points not included in the dataset.

For instance, if one were to build a model that accurately predicted the test scores that would be received by students based on a range of parameters (perhaps their attendance rates, number of assignments submitted, class participation and so forth) this model would provide all of the information relevant to predicting student scores in the dataset, and also information from arbitrary students or even imagined students who do not exist in the real world. If you could find a way of communicating

your model that was simpler than the dataset, you could regard this simpler model as a compressed form of the dataset, since all information provided by the dataset is by definition provided by the model also, at least in theory.

Of course, realistically, few models will generalise so perfectly, and so the model will usually only encode part of the required information. In the case of the test example, this can be resolved by combining the model description with deltas documenting how far, for each data point in the dataset, each output from the model is from the true value. What is happening here is that the model is being combined with all of the other information a user would need to reconstruct the dataset. Since smaller deltas can be stored in fewer bits, more accurate models will still be preferred, conditional on the model itself also being small. Thus, the model is evaluated by adding the length of its description in bits to the length of the deltas in bits, with smaller numbers being preferred.

This concept can be formalised as follows – given a model M and a dataset D, the MDL of M and D is defined as the value of Equation 2.8.

$$MDL(M, D) = L(M) + L(D|M) \tag{2.8}$$

where $L(M)$ (called the *model complexity*) is the length of the model M (usually expressed in bits), and $L(D|M)$ is the length of the dataset *after compression by* M. Again, this is usually expressed in bits. The exact nature of both of these terms and how they are computed can vary

depending on the application, but the model will generally have some canonical representation (e.g. as a matrix of weights for a neural network) that can be used to determine the model complexity. The example of deltas from the real test results given above is a good example of how the dataset length after compression can be calculated.

Given a set of models H (for hypothesis), MDL can be used to determine which one is best for the dataset. This is done by finding the hypothesis/model that produces the smallest resulting MDL value, as in Equation 2.9.

$$Best(H, D) = \underset{M \in H}{arg\,min}\ MDL(M, D) \tag{2.9}$$

One purported advantage of MDL (Grunwald, 2004; Grünwald et al., 2005) is that it provides a way of evaluating a machine learning model that automatically accounts for overfitting. This is because overfitting requires a large, complex model. Thus, although overfitting to the dataset will produce a small value for $L(D|M)$, it should give a very large value for $L(M)$. In general, this means that models that are liable to overfit will be penalised.

2.5.1 The eGrids grammar induction algorithm

Given the background knowledge covered in the previous subsections, it will now be possible to describe the eGrids algorithm. There are three steps involved, so they will be covered in turn. The first step, which is an

initialisation step, consists of creating an initial grammar G. Each word observed in the dataset is given its own associated nonterminal, so for example if the word “likes" appears in the dataset, the nonterminal $LIKES$ will be created, and it will be assigned a rule $LIKES \mapsto "likes"$. Then, each sentence in the dataset is added as the right-hand side of the starting nonterminal S. For example, if the sentence *I like grammar induction* appears in the dataset, the rule $S \mapsto$ *I LIKE GRAMMAR INDUCTION* will be added to the rule set R.

Like in ADiOS, the remaining two steps are carried out iteratively. The second step is the most complicated, and involves the refinement of the existing grammar. This is done by applying a beam search algorithm over a set of three operators to produce a number of new grammars, which can then be evaluated. The three operators are described below in turn.

- **CreateNT** (create nonterminal) takes two nonterminals a and b and returns a new nonterminal α, defined by the rule $\alpha \mapsto ab$. All instances of ab in the grammar are then replaced by α.

- **MergeNT** takes two nonterminals a and b and returns a new nonterminal α defined by two rules $\alpha \mapsto rhs(a)$ and $\alpha \mapsto rhs(b)$, where $rhs(x)$ denotes the right-hand side of any rules the x nonterminal appears in. Since this can be more than one for both a and b, it actually adds a *minimum* of two rules, and could add many more, one for

each right-hand side associated with the nonterminals in question. The MergeNT operator can be clarified with an example. Suppose eGrids is working with a grammar consisting of the following rules:

- $S \mapsto A\ B$
- $A \mapsto C\ D$
- $B \mapsto E\ D$
- $C \mapsto F\ G$
- $D \mapsto G\ H$
- $E \mapsto D\ I$

And MergeNT is run on the B and D nonterminals, the following grammar would result:

- $S \mapsto A\ \alpha$
- $A \mapsto C\ \alpha$
- $\alpha \mapsto E\ \alpha$
- $\alpha \mapsto G\ H$
- $C \mapsto F\ G$
- $E \mapsto \alpha\ I$

- **CreateOptionalNT** differs from the other two operators because it only operates on a single rule. Given a rule $x \mapsto y$ as input, it adds a new rule $x \mapsto y\ z$ to the dataset, where the string "$x\ z$" is observed to have occurred in the input dataset at least once.

The final step of the eGrids algorithm is to evaluate the grammars produced by the operators, and to see which (if any) are an improvement on the previous grammar. This is where MDL comes in. Petasis et al. (2004) state that the evaluation metric for each grammar G is computed as in Equation 2.10.

$$Eval(G) = GDL(G) + DDL(G) \tag{2.10}$$

Here, GDL stands for *grammar description length*, and DDL for *derivations description length*. These are equivalent to the $L(M)$ and $L(D|M)$ terms from the MDL equation (see Equation 2.8, above). The authors of the paper present a binary encoding scheme for the grammars, to allow the computation of the GDL term, and a binary encoding scheme for the parts of the dataset that cannot be parsed by the grammar G to compute the DDL term. This is an example of MDL being used in an explicit manner.

If any of the grammars evaluated are better than the grammar from the last iteration, then stages two and three are repeated on this better grammar. If no better grammar exists, then the algorithm halts and the grammar from the last iteration is returned as output.

2.6 Other approaches to activity discovery

Apart from the topic models presented in Section 2.2, a number of other approaches to activity discovery have also been documented in the literature. This section will provide an overview of a number of these algorithms which have shown promising results.

Cook et al. (2013) presents a particularly interesting example, which uses an MDL-inspired algorithm to carry out activity discovery. This approach uses a greedy beam search to identify patterns that can minimise the description length of the dataset compressed by means of the pattern. An operator called *ExtendSequence* is used to add event types to candidate activities. The discovery process is initialised by placing each activity type into a candidate activity of its own. Each candidate activity is then extended using an operation called *ExtendSequence*, which adds new events to the activity using events that are observed to have occurred before or after instances of the activities observed in the dataset. An MDL-inspired metric is then used to decide whether an extension output by *ExtendSequence* is accepted. Specifically, the system always greedily accepts the activity a such that the full set of activities M (including a) maximise the ratio between the size of the dataset (written as $L(D)$) and the MDL equation, as shown in Equation 2.11. Cook et al. (2013) call this evaluation operation *EvaluatePattern*. A pseudocode representation of the basic process is presented in Algorithm 2.

$$EvaluatePattern(M, D) = \frac{L(D)}{MDL(M, D)} \tag{2.11}$$

```
activities = [item for item in set(D)]
while not finished:
    extended_activities = []
    scores = []
    for activity in activities:
        extended = ExtendSequence(n, activity, D)
        test_dataset = activities[:].replace(
            activity,
            extended
        )
        evaluation = EvaluatePattern(test_dataset, D)
        extended_activities.append(extended)
        scores.append(evaluation)
    best_index = scores.index(max(scores))
    activities.pop(best_index)
    activities.append(extended_activities[best_index])
```

Algorithm 2: The activity discovery algorithm proposed by Cook et al. (2013). After initialising by placing each event type into its own activity, a beam search algorithm is used to find activities by combining sets of events which maximise $EvaluatePattern$ score.

Another approach to activity discovery called the *Genetic Algorithm for Interleaved Sequences* (GAIS) was presented by Ruotsalainen et al. (2007). As the name suggests, GAIS is a genetic algorithm, which evolves a partition over a dataset of events E. Each event, $e \in E$, is assumed to be a tuple consisting of an event type and a time. The algorithm maintains a set of candidate solutions (chromosomes, to use the terminology typically employed for genetic algorithms), each of which consists of N partitions, and each partition is assumed to represent a

single activity. The fitness of a particular partition P_k of length n (each element of which is a sequence of events B_i) is computed by the fitness function f, defined in Equation 2.12. Note that the algorithm attempts to find a partition that *minimises* this fitness function.

$$f(P_k, M) = \sum_{i=1}^{n} \alpha_i \min_j (Edit(B_i, m_j) + L_1(B_i, m_j)) \tag{2.12}$$

where M is a database of *sequential model patterns*, each one of which (m_j are assumed to be representative of realistic activities that are expected to be found in the dataset. The function $Edit$ is an edit distance algorithm, while the function L_1 is the Manhattan distance. Therefore, the fitness function rewards partitions of the dataset that are similar to the example activities in the model patterns database. The weight α_i is equal to 1 for the largest distance in the partition, 2 for the second largest distance, and so on up to n for the largest distance. This is intended to prevent candidate partitions that consist of a mix of activities of varying qualities from being prematurely rejected by the fitness function.

One interesting aspect of GAIS is how the two major operations used in a genetic algorithm (crossover and mutation) are achieved on a partition of a dataset. For the crossover operation, a histogram of event types is constructed for each activity, and a *crossover point is selected within this histogram*. New candidate partitions can then be built by moving

events from the dataset into individual activities such that the makeup of each activity matches the new histogram for that activity. Mutation is carried out by moving events between activities within a single candidate partition. Two types are employed: a *swap mutation* finds two events of the same type in two separate activities and swaps them around, while *relocation mutation* simply moves an event from one activity to another.

GAIS is significant from the perspective presented in this thesis because it is a rare example of another activity discovery system which aims to address the interleaving problem. However, a significant weakness of this approach is its dependence on a database of model patterns, which requires significant prior knowledge of the problem domain that the sensor events are generated from. This significantly reduces usefulness of this algorithm.

Aztiria et al. (2012) present an ambitious system called the *Patterns of User Behaviour System* (PUBS). PUBS consists of an internal association mining algorithm $\mathcal{A}_{PUBS}$, which tries to find associations between sensor types within the dataset. In particular, it tries to identify temporal relationships between sensor types (i.e. sensor A seems to activate frequently within n seconds of sensor B), and contextual conditions under which the associations are observed to occur (for example, getting up after the alarm clock rings only happens on weekdays, not on weekends). One example given by Aztiria et al. (2012) of the types of associations that PUBS can find successfully is observing that *the bathroom fan is*

turned on within 10 seconds of turning off the shower if the humidity level in the bathroom is above 75%.

Interestingly, PUBS produces output in the form of a human-readable language, which is called $\mathcal{L}_{PUBS}$. The scenario involving the bathroom fan discussed in the previous paragraph can be represented in this language is presented as Algorithm 3. The final part of PUBS is a user interface component, $\mathcal{I}_{PUBS}$, which allows users to view, refine and delete the patterns discovered by the system.

```
ON occurs(Shower, OFF, t0)
IN context(Bathroom humidity level, (>, 75\%))
THEN do(On, BathFan, t) when t = t0 + 10s
```

Algorithm 3: An example of an association for the scenario *the bathroom fan is turned on within 10 seconds of turning off the shower if the humidity level in the bathroom is above 75%*, written in the $\mathcal{L}_{PUBS}$ language. Taken from Aztiria et al. (2012)

PUBS has an impressive ability to detect very complex relationships between different sensor event types. However, it is not a true activity discovery system, as the associations it finds are generally too low-level to serve as true activities. In spite of this, the idea of thinking in terms of *relationships* between sensor events is one which will be returned to later in this thesis.

Unbound Unsupervised Activity Discovery using the Temporal Behaviour Assumption (UnADevs) is an activity discovery system developed by Gjoreski and Roggen (2017). UnADevs takes real-valued inputs, and a sliding window is run over these inputs and fed as input to a clustering process, which forms the core of the UnADevs system. The clustering

process maintains a pool of $activePool$ *active* clusters, which are candidates to be output as discovered activities by the system. Active clusters can overlap with each other, so that alternative clusters for the input vector can be considered by the system. Each sliding window is converted into a feature vector, which is compared to the existing active pool of clusters by means of a Gaussian kernel distance metric. Each window is then placed into its own cluster, and into the closest matching existing cluster. If the number of clusters active exceeds $activePool$, the two most similar clusters are merged together. Once a cluster has not been updated for a period of time (the time being a user-selected parameter called the *tolerance*), it can be output as a discovered activity by the system.

The final activity discovery system that that will discussed in this section is that which was proposed by Safavi et al. (2020). This system is unique, in that it tries to discover activities in the form of representations of entities which have connections to one another represented as a graph structure. For example, assuming that a graph structure was to be built from a person's emails. Each vertex on the graph could correspond to an email-related entity such as a contact, a file or an individual email. These entities could be connected to each other based on relationships observed in the dataset – an edge between two email nodes could represent replies, edges between email and contact nodes could represent authorship or receivership, and edges between file and email nodes could represent

email attachments. The aim of the model is to find groups of graph vertices that correspond to activities. Groups of email threads about similar topics could be an example of the type of activity one would expect this system to find.

The model uses a form of graph propagation algorithm to learn the activities from the graph models given to it. The output of the model is an entity-attribute relation matrix. Each row of this matrix can be understood as an embedding-like representation of each a single vertex in the graph. The embeddings are seeded by noun phrase and topic frequencies found in files and emails in the dataset, and are iteratively updated by algorithm using a graph propagation algorithm. The graph propagation is done with the intention of minimising the loss function shown as Equation 2.13.

$$\mathcal{L}(\hat{X}) = |\hat{X} - X| + \lambda tr(\hat{X}^T L \hat{X}) \tag{2.13}$$

where $\hat{X}$ is the entity-attribute relation matrix to be evaluated, X is the initial entity-attribute relation matrix which seeded the algorithm, λ is a user-defined hyperparameter, L is the Laplacian matrix of $\hat{X}$, and tr is the linear algebraic trace function (i.e. it sums the main diagonal of its input matrix).

This approach is highly novel, and shows an example of activity discovery being used in a domain that is quite different from the type as-

sumed in this work.

2.7 Formal definition of activity discovery

With a number of concrete examples of activity discovery having been given, it is now possible to introduce a formal definition of activity discovery which will be used throughout the remainder of the thesis.

Formally, one can model an activity discovery system as a tuple consisting of six elements, (Σ, D, A, X, f, g), where:

- Σ is a set of *event types*;
- D is an ordered sequence of events, $D = \langle d_1, d_2, \ldots, d_L \rangle$ of length L, such that each $d_i \in D$ is drawn from the set Σ. This is called the *dataset*;
- A is a set of *activity types*;
- $X \subset \Sigma^*$ is the set of possible sub-sequences of the dataset D;
- f is a mapping $f : D \rightarrow X^*$, which takes a sequence of events D as input, and returns a set of (possibly non-contiguous) sub-sequences of D as output; and
- g is a mapping $g : X \rightarrow A$, which takes a sub-sequence produced by f as input, and returns an activity type $a \in A$ as output.

Multiple similar activities can then be clustered or lifted into one *type*. This means, for example, that if the activity discovery system observes

an activity occurring in the kitchen nightly, it may cluster them all into a single *making dinner* activity type. In the formalism above, A denotes the set of all such activity types. The concrete sub-sequences of D are referred to as the *instances* of the *making dinner* activity, and constitute a sub-set of X^*.

Note that it is not expected that an activity discovery system can operate with human-like semantic knowledge or expectations in the basic case. Thus, it would not be expected to be able to name the new activity type as *making dinner*, only to identify that the instances involved can be sensibly clustered together. An activity discovery system deployed in a real-world environment might well be supplemented with real-world knowledge, with the intention of biasing towards the sort of activities one would expect to find in the environment in which it operates. For instance, knowledge that events relating to a fridge or oven indicates activities relating to food preparation such as *making dinner* are taking place. In many ways this would stray over into being a form of activity recognition as well as discovery. For this reason, this sticks to a pure form of activity discovery without any real-world knowledge. However, it would still be expected to be able to discover *making dinner* as an activity, just not to be able to give it a label (*making dinner*) that would be semantically meaningful to a human observer.

2.8 Datasets

Like all machine learning research, activity discovery depends on the use of standardised datasets which allow the performance of various systems to be directly compared to each other. Ideally, these datasets used should have a number of characteristics:

- They should be *realistic*, in that they should closely emulate the inputs that would be expected to be provided to an activity discovery system in the real-world.
- The contents of the dataset should consist of interleaved activities, since interleaving is a major topic of the research presented in this thesis.
- The length of the dataset should be large enough to provide adequate training data for training and testing purposes.
- The data should be freely available – datasets that are only available to a select group of researchers, or that require the signing on non-disclosure agreements before access is granted should be avoided.

Fortunately, a number of datasets that meet these requirements exist. Many of them were initially gathered for use in the field of activity recognition, but can be used unmodified for activity discovery. One of the more commonly know of these is the van Kasteren dataset (Van Kasteren et al., 2008). This dataset consists of sensor readings gathered by sensors

placed around an apartment occupied by a single person for a period of one month. Although well know in the literature, the van Kasteren dataset does not contain a great deal of interleaving, which makes it unsuitable for the research presented here.

Another well established dataset is the *SCARE dataset* (Stoia et al., 2008), which is a corpus of situated dialogues, where activities were carried out in a virtual environment (see Figure 2.5 for an example) by teams of two volunteers. One volunteer controlled movement and action in the virtual environment and one directed this controlling volunteer by providing them with instructions on how to complete a list of given tasks. Each task involved moving an object in the environment from one location to another.

The *TU Munich kitchen dataset* (Tenorth et al., 2009) provides a dataset gathered in a kitchen environment, rather than a full house/apartment as in the van Kasteren dataset. This makes it more limited than some of the other datasets, which should ideally try to gather data representing a wide range of activities that occur throughout a house. Another limitation of the TU Munich dataset is that much of it is video-based – although it does have a number of simple, discrete-valued sensors, it is primarily based on video footage. Producing events that could be fed into an event-based system such as those that this thesis focuses on is possible in theory, but is beyond the scope of this thesis.

More relevant to the work presented in this thesis is the *Kyoto 3 data-*

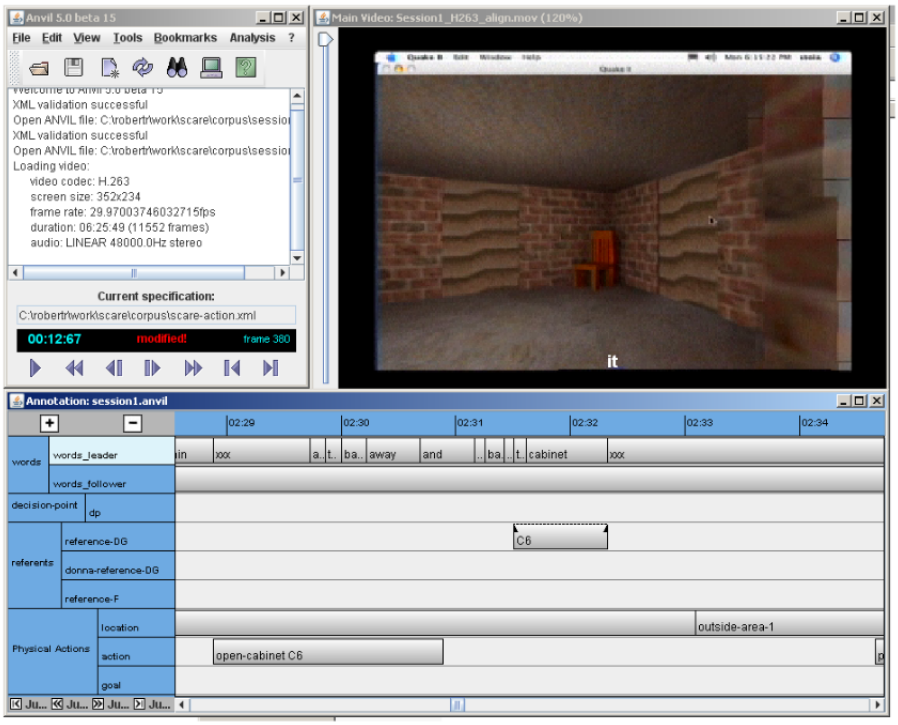


Figure 2.5
An example of the 3D virtual environment used by the SCARE corpus, displayed above an annotation of the events taking place. The software package shown, which was used by Stoia et al. (2008) to align the annotations and video, is called *Anvil*. Image taken from Ross and Kelleher (2013).

set gathered by the CASAS project (Cook and Schmitter-Edgecombe, 2009). Like van Kasteren, this dataset consists of readings from a range of sensors installed in a small apartment, but contains explicitly interleaved activities. The dataset was gathered by asking a number of participants to perform *activities of daily living* (ADLs) in a natural manner in the apartment.

Some of the research published previously in the field (such as the work by Huynh et al. (2008), mentioned in Section 2.2) utilises datasets gathered by the authors of the published work. Unfortunately, these datasets are often not available to the public, even though they would be

useful to allow direct comparisons with existing systems.

Finally, the *Opportunity dataset* (Roggen et al., 2010; Sagha et al., 2011) is considerably more complex than the other datasets covered so far. Much like the Kyoto dataset, the Opportunity dataset was gathered by having human volunteers carry out a range of activities in a mock studio apartment. The apartment was fitted with a range of sensors to detect the actions of the volunteers – sensors to detect when drawers or cupboards were opened or closed, accelerometers attached to kitchen utensils to detect their orientation and movement, and sensors to record when an appliance was switched on or off. In addition, on-body sensors were worn by the volunteers to record the motion of their limbs. Unlike some of the other datasets used here, the Opportunity dataset uses real-valued sensors almost exclusively. Like Kyoto, this dataset takes place in the real-world, not in a virtual environment as was the case for SCARE. Opportunity is interesting in that its ground truth consists of both high-level activity labels and lower level *gestures*. One persistant issue addressed in this thesis (and covered in particular in Chapter 7) is that of how to evaluate activities at differing levels of abstraction. Although far from perfect, the Opportunity dataset's labelling is a step in the right direction towards abstraction-aware, hierarchical labelling of ground truths.

2.8.1 Datasets used in this thesis

Three corpora are used in various points in this thesis. The primary dataset used was the SCARE corpus, since it is both lightweight and interleaved, but both the Kyoto 3 and Opportunity datasets (which are much more heavyweight interleaved datasets) were also employed to more thoroughly test the language model systems presented in Chapters 4 to 6.

Since the SCARE corpus was collected for research into situated dialogues, it had to be converted into a format more suitable for activity discovery, as in Ross and Kelleher (2013). This was done by processing the full dataset into a list of *events* in the virtual environment. Whenever some aspect of the environment changes (for example, when a door is opened, or when an object is picked up or placed down), a new event is added to this list list. The completed list therefore presents the events that took place in the temporal order in which they occurred in the original dataset. An extract of the resulting dataset is shown below:

$$Open\_Door\_D8, Open\_Cabinet\_C7, Close\_Cabinet\_C7, ... \quad (2.14)$$

This results in a dataset with five types of activities present in the ground truth. Although each activity occurs exactly 15 times in the dataset, the average *length* of each activity instance varies quite considerably between activity types (some had an average length of over 30 events,

Table 2.1
Average length of the various activities present in the SCARE dataset

Activity name	Number of instances	Total events	Average events
Move Quad	15	371	24.733
Move Picture	15	88	5.867
Move Rebreather	15	566	37.733
Move Silencer	15	474	31.6
Move Box	15	104	6.933

some had an average length of less than 7). This makes the SCARE dataset quite *imbalanced*, which could present a challenge to any activity discovery model. A list of the activities present in the dataset is shown in Table 2.1. For each activity, the table shows the number of instances of the activity (15 for each, in this case), the total number of events present in the dataset that are instances of each activity type, and the average number of events for each instance (i.e. the total divided by 15).

About 15% of the dataset consisted of interleaved activities, and about 73% of just one goal being active at a time.

A similar preprocessing stage was employed for the Kyoto 3 dataset. Most of the sensor readings are either binary (they have a simple on/off state), or can only enter one of a handful of states, as shown in Figure 2.6. This means they can be easily converted to the sequence-of-events format the proposed systems expect by creating event types of the form $SensorName\_SensorState$. For example, one of the sensors is referred to as $M17$ in the dataset, and can take the state ON, so $M17\_ON$ becomes an event type in the dataset. For the few sensor types that did

2008-07-29	13:05:16.438795	M07	OFF	1
2008-07-29	13:05:16.438795	M08	OFF	1
2008-07-29	13:05:16.438795	M09	OFF	1
2008-07-29	13:05:19.430101	M14	ON	1
2008-07-29	13:05:19.430101	M15	ON	1
2008-07-29	13:05:19.679056	M16	ON	1
2008-07-29	13:05:20.38868	M14	OFF	1
2008-07-29	13:05:22.609089	M17	ON	1
2008-07-29	13:05:22.609089	M15	OFF	1
2008-07-29	13:05:22.609089	D07	OPEN	1
2008-07-29	13:05:25.524211	M17	OFF	1
2008-07-29	13:05:25.524211	M16	OFF	1

Figure 2.6
An extract of the Kyoto 3 dataset prior to preprocessing. The columns for each event are respectively date, time, sensor name, sensor state and ground event ID.

have continuous values, the *Jenks natural breaks algorithm* (Jenks, 1967) was used to discretise the data.

Thus, the sequence of sensor events show in Figure 2.6 would be converted as follows after preprocessing:

$$M07\_OFF, M08\_OFF, M09\_OFF, M14\_ON, ... \tag{2.15}$$

An overview showing the activity types present in the dataset, and information on their frequency and average length, is shown as Table 2.2.

The Opportunity dataset mostly uses real-valued sensors, so the preprocessing was more computationally expensive for it than for the other datasets presented. Again, the Jenks natural breaks algorithm was used

Table 2.2
Average length of the various activities present in the Kyoto 3 dataset

Activity name	Number of instances	Total events	Average events
Fill-Medicine	37	633	17.11
Prepare-Soup	48	1251	26.06
Water-Plants	26	1096	42.15
Choose-Outfit	23	564	24.52
Clean	36	1778	49.39
Watch-DVD	44	1100	25.00
Prepare-Card	28	892	31.86
Answer-Phone	24	494	20.58

to discretise the data and convert it into events in the same way as was done for the real-valued sensors present in the Kyoto dataset. As for the other datasets, statistics for Opportunity are presented as Table 2.3.

2.9 Discussion

The numerous approaches to activity discovery covered in this chapter (topic modelling, motif discovery, grammar induction, MDL-based approaches, GAIS, PUBS, UnADevs and the system presented by Safavi et al. (2020)) can all be said to provide real-world implementations of the formal description given in Section 2.7.

Topic models are still widely used for activity discovery (Zhao et al. (2020) is a recent example), and seem to be quite a reliable approach, with F-measures commonly in the 70 to 80% range. It is worth noting, however, that Zhao et al. (2020) use a pseudonymised public transport dataset which does not appear to be interleaved. Motif discovery al-

gorithms tend to be somewhat less common, which is surprising, since they are arguably similar to topic models in terms of what they are trying to achieve. However, the nature of the discussed planted motif problem seems somewhat contrived. In the case being considered in this thesis, it is very unlikely that an activity will always fit within a fixed length l. A better way of understanding activities is that they are sequences of actions (or perhaps sequences of both mandatory and optional actions) where each action can be interleaved with both other actions and with noise to varying degrees.

In spite of this, one interesting (and potentially useful) aspect of motif discovery, which is not shared by topic models, is that most of them have an explicit awareness that there will be parts of the dataset that are *not* part of any motif. Topic models will try to classify every document they are given into one or more topics. Since the dataset is split into documents (typically using sliding windows, as discussed in Section 2.2), a topic model will attempt to classify *every* window into a topic. This includes windows that are not part of any activity, and thus many of the documents might end up being classified into spurious topics that do not correspond to any activity.

Grammar induction suffers from a very similar problem. Written sentences are of course not interleaved. One could argue that spoken sentences can contain interjections, which are similar to having events that are not part of any activity in an activity discovery context. However,

interjections can be quite rare (often only constituting a tiny minority of words, while non-activity events could well be the majority of a dataset), and are often semantically valid parts of a sentence, expressing things like emotional reactions, which does not apply to non-activity events. In some ways, grammar induction is the *least similar* of the presented approaches to activity discovery. It is still a useful source of inspiration, however, and it will be argued later in this thesis that the output format of grammar induction algorithms (parse-tree like structures) might be a better format for activity discovery systems to aim for, rather than linear regions of activities as is currently common.

MDL is not really a class of algorithm or computational problem in the same sense as the other approaches presented in this chapter. A huge range of techniques could fall under the MDL banner, and one could develop an activity discovery system that uses MDL along with any of the other approaches discussed. MDL is more of a tool for evaluating and developing machine learning models (both the eGrids system presented in Subsection 2.5.1, and the system proposed by Cook et al. (2013), and described at the start of Section 2.6, are perfect examples of the type of thinking involved) which emphasises tradeoffs between increased accuracy and model simplicity. Given the recent trends towards increasing complexity in machine learning, in part due to recent progress in fields like deep learning, this might be a good thing to keep in mind. Thus, minimum description length will be mentioned a number of times in

future chapters, where it will be argued that it can serve as the basis for better evaluation metrics than are generally used in the field.

As noted in Section 2.6, the GAIS algorithm is significant because it is also explicitly aimed at dealing with interleaving, although it does so in way that appears to be quite dependent on having prior domain knowledge. Approaches to activity discovery that both explicitly address interleaving, while also assuming no prior domain knowledge, appear to be rarely investigated in the literature. The system proposed by Safavi et al. (2020) is novel, but it assumes access to a graph-like structure representing relationships between the entities being considered by the activity discovery system. This is not something that applies to the framework discussed in this thesis, since it is assumed that the input simply consists of a stream of sensor events, with no richer structure present.

By contrast, the PUBS and UnADevs systems are highly relevant to the work that is presented in this thesis. Although the approach that will be presented in later chapters is very different to either PUBS or UnADevs, it does share obvious similarities. PUBS is based on trying to identify relationships between sensors and sensor event types that can be observed from the dataset. This is a useful intuition, and one that will be built upon in future chapters. Likewise, this thesis will again return to the idea of using clustering algorithms to convert activity instances into activity types.

Table 2.3
Average length of the various activities present in the Opportunity dataset

Activity name	Number of instances	Total events	Average events
Other	961	160695	167.22
Stand	1710	345239	201.89
Walk	1733	199213	114.95
Sit	169	136852	809.78
Lie	40	25395	634.88

2.10 Summary

This concludes this discussion of existing approaches to finding structure in sequential data in the literature. Of the four broad approaches outlined (topic modeling, motif discovery, grammar induction and MDL), all of them will have a degree of relevance for the remainder of the thesis.

Some of the strengths and weaknesses of these approaches were discussed in in Section 2.9. This sets up an understanding of the issues that face existing activity discovery systems, which will be addressed in the coming chapters.

Chapter 3

Topic Modeling Algorithms for Activity Discovery

In section 2.2, the idea of using topic models as the basis for an activity discovery system was introduced. In particular, systems proposed by Huynh et al. (2008) and Seiter et al. (2014) were presented, which use topic models to produce activities from “documents" consisting of lower-level events. The work presented in this chapter identifies and tries to address some of the limitations of these systems – particularly when looking at a dataset with a high degree of interleaving. It begins by introducing the topic model used, followed by describing the system and the differences between it and the earlier Huynh et al. and Seiter et al. systems. This chapter is based on work that has been published as Rogers et al. (2016).

3.1 Latent Dirichlet allocation

A number of approaches to topic modeling have been presented in the literature. One of the most common, and the approach used both in this chapter and by Huynh et al. (2008) and Seiter et al. (2014), is *latent Dirichlet allocation* (LDA). First proposed by Blei et al. (2003), LDA is a generative statistical model of a text corpus: in other words, it presents a (contrived) statistical model of how a corpus of text documents came to be. The term *Dirichlet* refers to a *Dirichlet distribution*, a family of multivariate probability distributions (*multivariate* in this case referring to a distribution over vectors rather than scalar values). A Dirichlet distribution is denoted $Dir(\alpha)$, where α is a positive, real-valued vector of length K. A sample $\Theta \sim Dir(\alpha)$ is another positive real-valued vector of length K, each element of which, $\Theta_i \in \mathbb{R}$, is a normalised value drawn from a beta distribution. The i$^{\text{th}}$ element of the α vector, α_i, is used as the distribution's α parameter. The probability density function of the Dirichlet distribution is defined as:

$$f(\Theta, \alpha) = \frac{\prod_{i=1}^{K} \Theta_i^{\alpha_i - 1}}{B(\alpha)} \tag{3.1}$$

where $B(\alpha)$ is the multivariate beta function:

$$B(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)} \tag{3.2}$$

and $\Gamma(n) = (n-1)!$ is the Gamma function.

A corpus C is then defined as a set of m *documents*, each of which is a sequence of words $\langle w_1, w_2, \ldots, w_n \rangle$, drawn from a vocabulary $w_i \in V$. LDA assumes that each document is generated according to the process shown in Algorithm 4.

$\Theta \sim Dir(\alpha)$
for document **in** 1 .. M:
    $N \sim Poisson(\xi)$
    for i **in** 1 .. N:
        $\mathrm{z}_i \sim Multinomial(\Theta)$
        $\mathrm{w}_i \sim p(w_i|z_i)$

Algorithm 4: The generative model used by latent Dirichlet allocation

Thus, the first step is to sample a vector Θ from a Dirichlet distribution representing the mix of topics in this document. The α parameter is supplied by the user, and its length K denotes the number of topics. The word length of the document, N, is generated by a Poisson distribution, with the parameter ξ again being supplied by the user. For each word w_i, a topic is chosen for the word using a multinomial distribution paramaterised over the Θ produced by the Dirichlet distribution. Finally, a word is selected using a distribution conditioned on the topic z_i. For example, if the topic was something like sports, one would expect words like *football* or *score* to have a higher conditional probability than would be the case for other topics.

A representation of the above model in *plate notation* is presented as Figure 3.1. Here the circles represent random variables, and the arrows

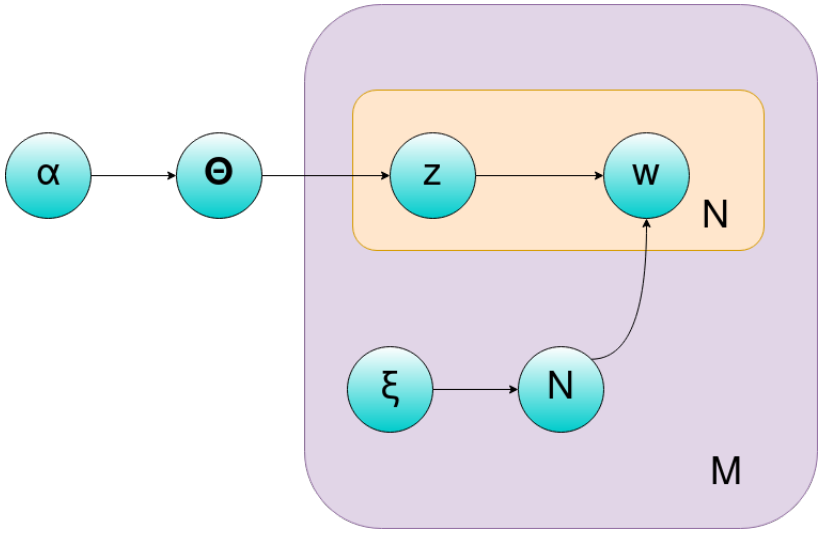


Figure 3.1
A plate notation diagram of the LDA model. The vector Θ is generated using a Dirichlet distribution parameterised over α. For each of the M documents, a document length N is generated, and then each individual word is generated by producing a vector z over topics and finally pick the word w based on the distribution of topics in z.

represent generative dependencies. Thus, one can see that the variable z has values generated based on the value of the variable Θ. This corresponds to the pseudocode above, which shows that each z_i is generated by a multinomial distribution that uses the vector Θ as its input parameter. The rectangular regions on the diagram indicate repetition of the generative process contained within them. The number of iterations is determined by the value shown on the bottom right of the rectangle: for each document $document_k$, the model generates N, which determines the number of words in the document. For each of the N words, the model generates a vector z_i over topics and a word w_i.

In order to use this model for topic modeling (i.e. to compute likely topics from given documents), the model must be *inverted*. This means taking the documents and a number of topics (K), both supplied by the

user, and fitting the model parameters to maximise the observations. This gives a probability distribution over topics for each document (actually for *each word* of each document, but analysing the output on such a low level may not be useful). *Gibbs sampling* is commonly used to do this, which involves searching over the space one parameter at a time, while keeping all other parameters fixed, to find the value which maximises the probability of the observations. Once one parameter is found, the algorithm moves on to find the value of the next parameter. Because changing other parameters may mean the first parameter is no longer optimal, this process occurs iteratively until the parameters converge upon the correct answer.

The topic modelling-based AD systems by Huynh et al. (2008), Seiter et al. (2014) and Rogers et al. (2016) presented in this thesis draw an analogy between the concept of a topic in natural language processing, and the concept of an activity. In the next section, the approach first published as Rogers et al. (2016) will be presented, and compared to Huynh et al. (2008) and Seiter et al. (2014).

3.2 Activity discovery with LDA

All existing topic modelling-base approaches to activity discovery are based upon an underlying intuition, which will be explained first before detailing the specifics of systems proposed by authors like Huynh et al.

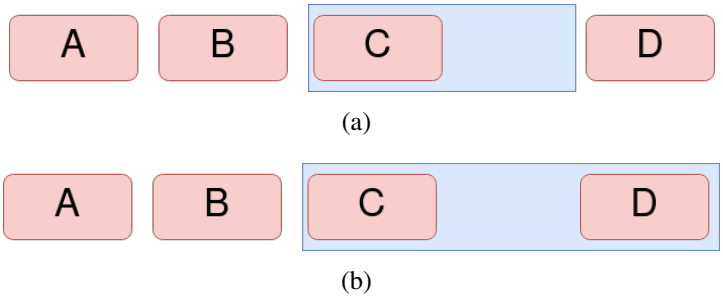


Figure 3.2
Comparison between the temporal and non-temporal approaches to sliding windows. Events C and D have a large gap between them, indicating that a large period of time passes between them. When a *temporal* sliding window is used, the *temporal period* covered by the window is fixed, so it will contain all events that took place within that period of time (Figure 3.2(a)). By contrast, when a *non-temporal* window is used (Figure 3.2(b)), all windows have the same length (2 events in this case), which causes the loss of information about the gap between events C and D.

(2008) and Seiter et al. (2014). Given a dataset D of L discrete events, a sliding window is run over the dataset to produce *documents*. The corpus of documents produced by this process is then used as input for a topic model, which classifies each document into a topic. Each topic is assumed to correspond to an activity.

The above process is used by almost all activity discovery systems that utilise topic modelling. One aspect by which such systems differ is how the length of the sliding window should be determined. Some sliding windows are simply defined as having a *fixed length* (in terms of the number of events), and thus all documents are the same size. Because the frequency of events can vary significantly, such windows can end up covering a varying temporal period. More commonly, however, the length of the sliding window is defined *temporally*. This means that a document is produced from all events that take place within a certain

window of time. For example, one might take all events that occur over a thirty second period and use them as a document. In this case, the temporal period covered by the window is fixed, but this means that the number of events can differ from one window to another, depending on the number of events that occurred in the temporal period in question. The window is then incremented by a certain *time quantum* – a short fixed period of time (say one or two seconds) – before going on to produce the next document.

The concrete system implemented by Huynh et al. (2008) is fairly similar to the temporal sliding window system described in the previous paragraph. However, one challenge faced by the authors was discretisation of the dataset: the dataset gathered was real-valued, and had to be somehow converted into discrete events to allow processing by the topic model. Initially, the authors simply carried out this discretisation process manually in order to verify that the LDA topic model worked, and that it could successfully discover plausible activities. Later on, they modified the system to add automatic discretisation. This was done by taking the raw, undiscretised dataset and running a temporal sliding window over it (much like the one later used to build documents for topic modelling), producing a set of real-valued vectors, each of which is the output from a sensor over a fixed-length time period. Another vector, of features extracted from each real-valued vector (the specific features used were generally statistical summaries such as the mean or variation

of the vector), was then produced for each window. These features were then clustered into discrete events, which were used as the input for the topic modelling step as described above.

Seiter et al. (2014) introduce a variant of the above algorithm that includes two novel variants of the basic LDA model. The first, called an *n-gram topic model* (NTM), adds n-grams regularly observed in the dataset to the vocabulary of events (the authors use terminology that differs slightly from that used in this thesis in that they refer to events as *activity primitives*.) A type of statistical significance test called a *permutation test* was used to identify n-grams from the dataset. Each n-gram was then treated as a new event type that could occur in the dataset and be fed into the topic model. For example, if *Event_A* and *Event_B* were seen to occur in sequence frequently in the dataset, the n-gram *(Event_A, Event_B)* was added to the vocabulary. Whenever *Event_A* and *Event_B* occurred together in one of the sliding window documents, *(Event_A, Event_B)* was added to the document before it was passed to the LDA model. The intuition behind this is that the events that occur in an activity are probably not independent, and thus the correlation between them should be modelled somehow.

The second topic model variant proposed by Seiter et al., the *correlated topic model* (CTM), is also inspired by the same observation. LDA makes the assumption that each word in a document is independent, which the authors argue is untrue. CTM replaces the Dirichlet distribu-

tion with a normal distribution, removing this assumption.

Directly comparing the performance of the two papers cited here can be difficult, since they use different datasets and different numbers of topics (the k parameter), and Seiter et al. does not publish an F-measure, but they both achieve a raw accuracy of about 70 - 80% when using around $k = 10$ topics, with Huynh et al. reporting an F-measure of about 75%.

3.3 Approach taken

While the approaches above apply a well understood technique to the challenge of activity discovery, it can be argued that the models proposed have a number of limitations. First, it could be argued that the models proposed above may actually be more complex than necessary:

- There may be no great advantage to using the temporal sliding windows over the dataset, and simply having a fixed window length (as described at the start of Section 3.2, thus essentially discarding temporal data) works well enough in many circumstances. It also has the advantage of reducing the quantity of data required as input, which can help alleviate privacy concerns.

- Activity discovery can still work well if the sensors being used are simple binary-based sensors. This greatly reduces the quantity of pre-processing required, lessens some of the privacy concerns

inherent in gathering data from continuous sensors (which often record highly detailed information, like the movement of individual limbs), and makes the resulting datasets considerably smaller and easier to process.

Another issue with existing approaches is that few of them address the issue of *hierarchy within activities*. In many cases, an activity will be a complex aggregate of simpler activities – for example, a *cooking dinner* activity could consist of smaller activities such as *chopping vegetables* or *heating in oven*. Modelling this process by developing activity discovery systems that explicitly produce hierarchical output could thus improve the quality of the output produced by such systems.

By contrast to the systems discussed previously, the system proposed in this chapter also introduces a novel way of combining discovered activities into complex hierarchies of activities, which it is hoped will better capture the complex hierarchical nature of real-world activities.

However, before discussing the hierarchical aspects of the approach, this section will first discuss the non-hierarchical aspects of the system. Initially, the system was developed to operate using the same basic algorithm proposed by Huynh et al.. Figure 3.3(a) shows an example of a sliding window of length 3 over a dataset of length 5 (in reality, the subset of the SCARE corpus that was used as a dataset has a length of over 1600 events – see Section 3.4 for details of the dataset). In Figure 3.3(b) one can see the sliding window has moved one event forwards.

Unlike in the work of Huynh et al., this approach always increments the sliding window by exactly one event at a time, giving this system a larger number of documents to work with. Thus this has produced two documents, and this process continues to generate more documents until the window reaches the end of the dataset. Interestingly, it is also possible to produce a probability vector over topics/activities for each event. This is done by computing the sum of the probability distribution vectors for all windows/documents that contain the event, and then re-normalising the resulting vector to produce a probability vector over topics/activities. Distributions produced by this method were used to evaluate the system's performance, and they also provide the basis for the hierarchical analysis.

Formally, the presented system operates over three stages. First, it moves the sliding window (of length w) over the dataset to produce a set of documents $\boldsymbol{w}$. These are fed into the LDA topic model, thus associating each w_i with a vector over topics/activities, which will be denoted t_i. As noted above, it then computes the activity probability for each *individual* event in the dataset d_j (note that j is used to index over events in the entire dataset, and i to index over sliding windows). This was done by adding together all probability vectors in the subset of $\boldsymbol{t}$ containing d_j, which is denoted as $T(j)$, as show in Equation 3.3. Pseudocode for the process is also presented as Algorithm 5. The approach determines that a contiguous subset of the dataset belongs to an activity if they all

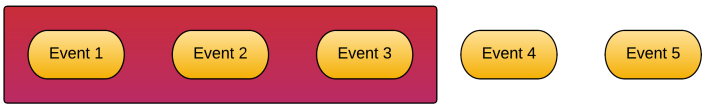
have been classified as having the same topic/activity by this algorithm, and the probability assigned to this topic/activity remains above a certain *threshold* for all of the contiguous subset.

$$d_j = \frac{\sum_{v \in T(j)} v}{|T(j)|} \tag{3.3}$$

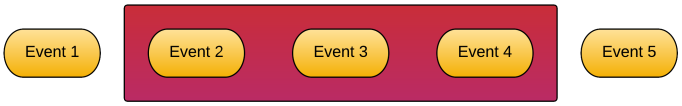
```
def compute_individual_probability(dataset,
                                   t,
                                   j,
                                   window_length):
    d_j = dataset[j]
    T(j) = [ t[i] for i in 0..|t|
             if d_j in dataset[i:i+window_length] ]
    return sum(T(j)) / |T(j)|
```

Algorithm 5: The algorithm used to compute the probability distribution over activities for a single event d_j

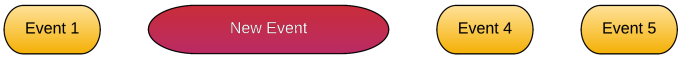
The hierarchical analysis aims to address the fact that although a pure topic modeling approach provides a somewhat robust means by which activities can be extracted from raw sensor data, it suffers from a substantial problem: it isn't clear what window size should be used when running the system. This is a user-defined parameter, but setting it to different values can result in profound differences in the resulting output. Worse, the resulting differences may not be simply "right" or "wrong", since as the window lengths increase the level of abstraction of the discovered topics will presumably increase. Many activities could plausibly have multiple levels of abstraction, and an activity discovered at any one level could still qualify as correct. For example, if processing a dataset that contained events from a kitchen, one could imagine discovering an



(a) A sliding window covering events 1 to 3



(b) A sliding window covering events 2 to 4



(c) Events 2 and 3 have been abstracted into a single event

Figure 3.3
An illustration of the internal operation of the approach presented in this chapter

activity at one level of abstraction that corresponded to “making tea". With a smaller window size, one might then find an overlapping activity corresponding to “boiling kettle" (since boiling kettle could be a constituent activity of making tea). Likewise, with a larger window size an overlapping activity called “making dinner" might then be found (again, making tea could be a constituent activity of making dinner).

In order to make progress in solving this issue, the model follows the principal outlined in Figure 3.3(c). Here, *Event 2* and *Event 3* were found with high probability to belong to an activity (the probability distribution vector over topics for that event discussed above has a probability exceeding a user-supplied threshold for at least one activity), and the highest probability activity for both matched. This means that both were then removed from the dataset, and replaced with a new event, which

abstracts away the activity that this topic is presumed to represent. This entails re-interpreting an activity as an event after the activity has been discovered by the topic model. The algorithm is then re-run for a second iteration using this new dataset, allowing for the inference of higher level activities. This means that by using a fairly small, fixed window length, the system can be allowed to explore and discover activities over multiple levels of abstraction. A variant of LDA, Hierarchical LDA (hLDA), exists (Blei et al., 2010) which already allows for hierarchies of topics to be learned. This system, however, is doing something conceptually different. Whereas hLDA builds a hierarchy of topics based on perceived semantic similarity (i.e. news about football and basketball could be abstracted into a "*sports*" topic), the system builds hierarchies based on constituency, i.e. where one topic (or activity in this case) is a subset of, or overlaps with, another topic.

3.4 Dataset

To investigate the performance of the system, the *SCARE corpus* (Stoia et al., 2008) was used, which has previously been discussed in Section 2.8. Note that most of the temporal information contained in the dataset is discarded before passing it to the topic model, and thus only the order is preserved: information about the time at which the events occurred is no longer present in the dataset when the model proposed in this chapter

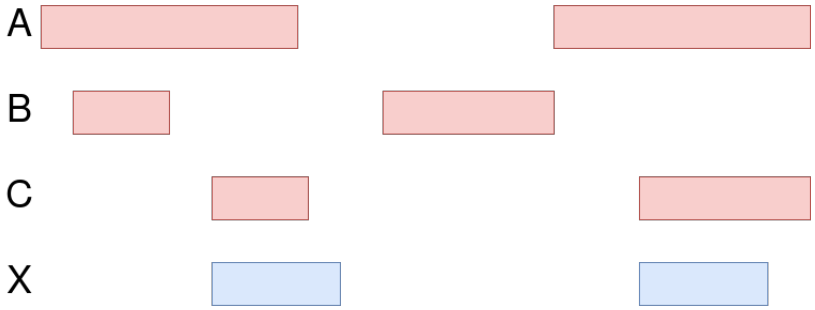


Figure 3.4
If A, B and C are ground truth activities, and X is an activity produced by an activity discovery system, which ground truth activity should X be compared to for evaluation purposes? Logically, it would seem to correlate best with activity C, so the evaluation metric will be run over C and X.

receives it, or even the amount of time that passed between individual events. This is because a non-temporal sliding window is being used, as discussed in Section 3.2.

3.5 Results

One major theme of this thesis is that evaluation of an activity discovery system is always a challenge (something that will be discussed in more detail later on, see Chapter 7 in particular). Usefully, the SCARE dataset contains ground truths, which can be compared to the system's output. Thus, F1 scores were used as an evaluation metric in this work, both the precision and recall of the system to be taken into account. In order for this to work, a way to match topics to their most probable corresponding label in the original dataset had to be found.

Figure 3.4 illustrates the intuition behind the solution to the problem of matching topics with ground truth labels. A greedy algorithm was

used for this – the pseudocode for this algorithm is presented below. The first step is to extract the most probable topic/activity for each event in the dataset using the topic modelling algorithm. The generated topics are then matched with ground truth labels by iterating over the ground truth labels. For each ground label, the F1 score is evaluated for each remaining topic, taking the current label to be the corresponding ground truth. One then assigns the topic with the highest corresponding F1 score to be the matching topic for the ground truth label. The topic and ground truth label is then removed from consideration before the algorithm repeats on the next label (i.e. this method will never assign a topic to more than one label).

The algorithm is reproduced below in pseudocode as Algorithm 6.

The results of running this evaluation over a window size of 22 (multiple window sizes were tested, and a window size of 22 was found to result in the highest mean F1 score) are shown in Table 3.1[1]. Some of the results obtained for other window lengths are also presented, namely for window lengths 10 (Table 3.2), 20 (Table 3.3), 25 (Table 3.4), 30 (Table 3.4) and 40 (Table 3.6). The relationship between window length and average (macro) F1 score is graphed in Figure 3.5. Due to the fact that the datasets being used are imbalanced (as previously shown in Table 2.1), it is possible to compute the average across all activity types either by weighting all activity types equally (macro average), or by weighting

[1]Note that for ease of reading, the ordering of the labels in the tables was preserved in order to be consistent with the ordering first seen in Table 2.1.

```
# Output to ground mapping
output_grnd_map = []
num_activities = len(output_activities)
while len(output_grnd_map) < num_activities:
    best_correlation = 0
    best_output = None
    best_ground = None
    for g in ground_truth:
        for o in output_activities:
            correlation = F1_score(g, o)
            if correlation ≥ best_correlation:
                best_correlation = correlation
                best_output = o
                best_ground = g
    out_grnd_map += [(
        best_output,
        best_ground
    )]
    g.remove(best_ground)
    o.remove(best_output)
```

Algorithm 6: The algorithm used to map output activities to ground label instances

Table 3.1
Performance of the LDA-based topic model system running with window length 22

Topic	Label	Precision	Recall	F1 score
Topic 0	Move Quad	0.7353	0.5556	0.6329
Topic 1	Move Picture	0.9780	0.1295	0.2288
Topic 2	Move Rebreather	0.8923	0.4947	0.6365
Topic 3	Move Silencer	0.8015	0.1870	0.3032
Topic 4	Move Box	0.0	0.0	0.0
Topic 5	None	0.0449	0.0270	0.0338
Macro Average		0.5753	0.2323	0.3059
Micro Average		0.7759	0.3657	0.4734

activity types by their frequency in the dataset (micro average). Both macro and micro averages are presented in the result tables.

At first glance, these results might seem to lag behind the 75% F1 score reported by Huynh et al. (2008). However, this is due to a number

Table 3.2
Performance of the LDA-based topic model system running with window length 10

Topic	Label	Precision	Recall	F1 score
Topic 0	Move Quad	0.1294	0.1322	0.1308
Topic 1	Move Picture	0.1702	0.0231	0.0406
Topic 2	Move Rebreather	0.1606	0.1029	0.1254
Topic 3	Move Silencer	0.1282	0.0340	0.0538
Topic 4	Move Box	0.0404	0.0702	0.0513
Topic 5	None	0.0095	0.0044	0.0060
Macro Average		0.1064	0.06113	0.068
Micro Average		0.1365	0.0828	0.096

Table 3.3
Performance of the LDA-based topic model system running with window length 20

Topic	Label	Precision	Recall	F1 score
Topic 0	Move Quad	0.6088	0.5250	0.5638
Topic 1	Move Picture	0.8098	0.1375	0.2351
Topic 2	Move Rebreather	0.9396	0.3740	0.5350
Topic 3	Move Silencer	0.8079	0.1784	0.2923
Topic 4	Move Box	0.0000	0.0000	0.0000
Topic 5	None	0.0376	0.0289	0.0327
Macro Average		0.534	0.2073	0.2765
Micro Average		0.756	0.3139	0.4187

Table 3.4
Performance of the LDA-based topic model system running with window length 25

Topic	Label	Precision	Recall	F1 score
Topic 0	Move Quad	0.5941	0.5156	0.5521
Topic 1	Move Picture	0.8059	0.0901	0.1621
Topic 2	Move Rebreather	0.5711	0.4591	0.5090
Topic 3	Move Silencer	0.6733	0.1212	0.2054
Topic 4	Move Box	0.0100	0.0000	0.0000
Topic 5	None	0.0313	0.0257	0.0282
Macro Average		0.4476	0.2019	0.2428
Micro Average		0.5831	0.3222	0.3771

Table 3.5
Performance of the LDA-based topic model system running with window length 30

Topic	Label	Precision	Recall	F1 score
Topic 0	Move Quad	0.3574	0.3200	0.3377
Topic 1	Move Picture	0.6337	0.0878	0.1542
Topic 2	Move Rebreather	0.5729	0.3028	0.3961
Topic 3	Move Silencer	0.4184	0.1043	0.1670
Topic 4	Move Box	0.0000	0.0000	0.0000
Topic 5	None	0.0259	0.0175	0.0209
Macro Average		0.3347	0.1387	0.1793
Micro Average		0.4435	0.2166	0.2759

Table 3.6
Performance of the LDA-based topic model system running with window length 40

Topic	Label	Precision	Recall	F1 score
Topic 0	Move Quad	0.1309	0.1256	0.1282
Topic 1	Move Picture	0.2347	0.0308	0.0545
Topic 2	Move Rebreather	0.2070	0.1088	0.1427
Topic 3	Move Silencer	0.1924	0.0303	0.0523
Topic 4	Move Box	0.0000	0.0000	0.0000
Topic 5	None	0.0083	0.0050	0.0062
Macro Average		0.1289	0.05008	0.06398
Micro Average		0.1732	0.0781	0.0985

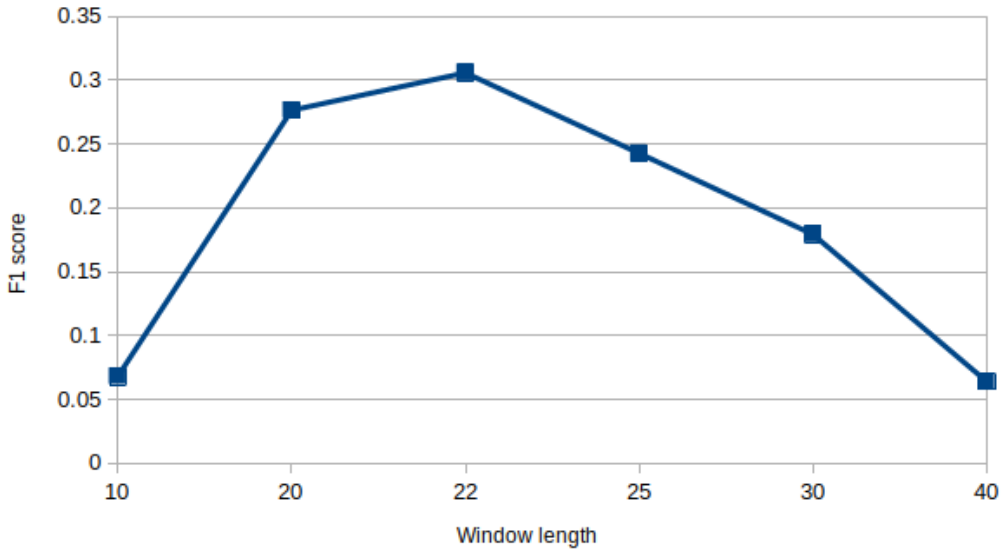


Figure 3.5
Relationship between window length (x-axis) and average F1 score (y-axis).

of important features of this work. To begin with, the dataset being used is fundamentally more difficult in two ways. Firstly, activities are interleaved in the SCARE dataset, and so the boundaries between activities are not as clear-cut as in the dataset used by other authors. Secondly, recall that the SCARE dataset uses binary sensors that are embedded in the environment, rather than on-body sensors producing complex motion data. This means that the system receives less information about what the user is actually doing, which is a significant handicap placed on it. However, one could argue that this is a good thing from a privacy perspective. Activity discovery (and recognition) systems always have to deal with a trade-off between better accuracy and more intrusive data collection, so approaches that can achieve reasonable performance with less data are generally to be welcomed.

Considering the results in more detail, one can see that the *Move Box* label seems to have been particularly difficult for the system to detect. Most instances of this activity are very short (about 6 sensor events long, compared to 20 - 30 for other activities), so it may not be a large enough pattern for the system to pick up on. The fifth topic was assigned to a label called *None*. This does not appear in the original SCARE dataset: there are 5 labels taken from the dataset, and a sixth label that was artificially added, which in effect means no activity was taking place.

Another problem with evaluating activity discovery systems is that there may not be a clean overlap between the natural patterns in the

data (which is what such a system might be expected to output) and the hand-annotated activities in the dataset itself, which are being used as ground truths for this evaluation. This indicates that the evaluation metric may in fact not be particularly well suited to this problem. It also provides no way of evaluating the hierarchical system that has been built. An alternative solution may simply be to visualise the patterns that have been detected, and then manually inspect them. Figure 3.6 illustrates an example of a diagram that was produced from the system's hierarchical output (the complete image is far too wide to reproduce here without loosing an unacceptable degree of detail). This image can be understood as a plot, with time plotted along the x-axis and the layers of the hierarchy along the y-axis. At the bottom of the image, different colours indicate different sensor events occurring over the course of time. Progressing upwards, towards the top of the image, one can see sensor events being replaced with new abstract events, corresponding to low-level activities as explained in the previous section. In the centre left of the image, an abstract event can be seen, which is created at an early stage in the hierarchy, and this event is subsumed into a new, higher level event about half-way up. At the top of the diagram, this new abstraction is in turn subsumed into an even higher one. The very right of the image shows an abstract event being created very late in the hierarchy that is never subsumed by anything. The late formation of abstractions is fairly common, and results from the relative probabilities of events that are

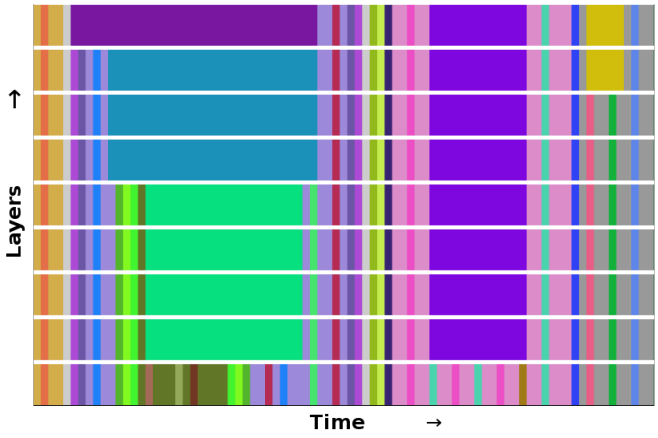


Figure 3.6
The diagram style begin used to visualise the topic modelling system's output. The passage of time is represented on the x-axis, while the y-axis shows the layers of discovered activities being built up. Each colour represents a distinct activity type.

carried over from previous layers changing due to the replacement of low-level events with abstract events. On top of this, repeating patterns of colour can be seen in the bottommost layer. These correspond to the activities that the system is producing.

A similarity can be noted between the hierarchical system that has been developed and the concept of maximising compression as the goal of pattern discovery, for example as used by (Cook et al., 2013). The topmost layer in the hierarchy is about 86.56% the length of the full dataset in the bottom layer.

3.6 Summary

The topic modelling based system that was in this chapter performs well in spite of the considerably less rich dataset provided as input than required by other approaches. No temporal information was provided to

the system for this approach, while authors like Huynh et al. (2008) and Seiter et al. (2014) convert the raw data stream into a format suitable for the topic model using a temporal sliding window, making those approaches inherently temporal. The work also used simple, binary sensor data, in constrast to the rich (real-valued) sensor data utilised by other authors, requiring an explicit discretisation step. This suggests that the approach presented here is a viable path for producing activity discovery systems in a less computationally costly manner. As ever, there is a trade-off involved: using the temporal sliding window systems gives a slight performance advantage, although at a cost of consuming greater computational resources, and a more complex system overall. The question of whether the extra computational load is worth the improved results can depend in the application, but this chapter has shown that the performance improvement is in many ways quite small.

This chapter also proposed the concept of building hierarchies of activities. It can be argued that activities are inherently hierarchical, and models built to be aware of this fact can more accurately capture the structure inherent in the data. This is quite an important contribution, in fact in many ways introducing this concept is the most important contribution of the chapter. Thus this hierarchical approach will be retained in all of the other systems presented later in this thesis.

Nonetheless, these activity discovery systems (and, indeed, most activity discovery systems proposed in the literature) have a number

of major weaknesses. Most notably, most of them deal poorly, or not at all, with interleaving. This is a topic that is rarely addressed in the existing literature, but the actions of people in the real-world are frequently interleaved. This implies that most activity discovery systems will have to learn to deal with the phenomenon to some extent, and the fact that the topic models cannot seem to do this is a major issue affecting current approaches.

A related problem involves the *ordering* of events in activities. In many cases the order in which events occur has a bearing on whether an activity can be said to be taking place or not. For example, when making a cup of tea or coffee, the kettle has to be boiled prior to pouring the water. Thus, if the sensor stream shows a person pouring water from the kettle before boiling it, this is a sign that they are *not* making tea or coffee, but must be doing something else instead. The bag-of-words approach used by topic models cannot take this into account – information on the order in which events were observed is ignored by the LDA model. It seems likely that this could be a significant handicap for existing activity discovery systems.

Activity discovery is a hard problem in general, and models like topic models might be *too simple* to deal with the complexity of information needed for accurate activity discovery. For this reason, this thesis proposes a completely novel approach to activity discovery which is not based on topic models at all. In the coming chapters, this approach will

be presented, which will be advocated as a compelling starting point for a new approach to activity discovery, and which addresses many of the issues raised above. In particular, this approach is explicitly designed to deal with interleaving, and to take the order of events into account.

Chapter 4

Neural Language Modelling for Activity Discovery

Chapter 3 introduced and evaluated an activity discovery system that utilises LDA topic models. However, the close of the chapter highlighted a number of limitations that affect topic models in general, and thus would lead to unsatisfactory performance in any activity discovery approach built on topic modelling. Two issues that particular attention was drawn to was the poor ability of topic models to deal with interleaved data, and the lack of any notion of event ordering in most topic models.

This chapter introduces the beginnings of a novel approach to activity discovery that can hopefully overcome these limitations in a way that existing approaches (including topic models) cannot. A key insight missed by other activity discovery systems is that fundamentally all such systems are trying to find a way to *model the relationship* between events in the dataset. For example, given a dataset $D = \langle A, B, C, D, E, F \rangle$,

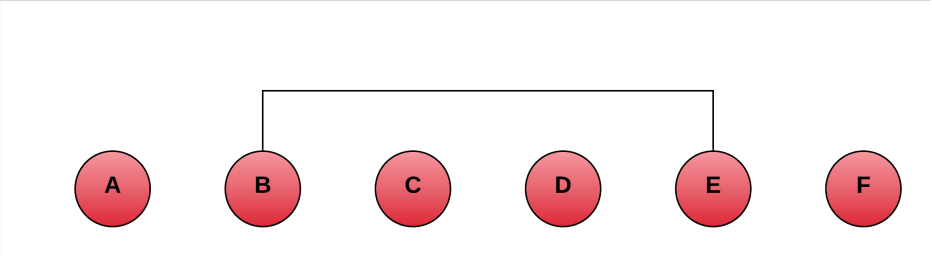


Figure 4.1
Linking events that seem to have a statistical relationship forms the essence of almost all activity discovery systems

and assuming that the presence of the event B allows the presence of the event E to be predicted (or vice versa), one can be confident that they belong to the same activity, as illustrated in Figure 4.1. This is true even if there are other events between them that cannot be predicted.

Thus, an activity discovery system could be constructed given some way to reliably predict events from other events. It is thus proposed that this can be done by means of a *neural language model*.

This chapter starts by introducing the concept of a language model in Section 4.1. A brief review of modern deep learning and neural network terminology is presented in Section 4.2, followed by a discussion of how deep learning models can be used to extract complex structure from sequential data in Section 4.3. This allows for a high-level overview of the approach being proposed in this chapter to be outlined in Section 4.4. In Section 4.5, the specific neural language model that was used to build the links described above is introduced. With all of the background material covered, Section 4.6 provides a detailed description of the approach. The chapter ends with an evaluation of the approach (Section 4.7) and a

summary of the chapter (Section 4.8).

4.1 Language modeling

A *language model* is a technique for computing probability distributions over sentences from a language. Thus, for example, a language model for English, would accept an English utterance (a sentence, paragraph, etc.) as input and output the probability that a native English speaker would speak or write such an utterance. In order to reduce the computational complexity involved, modern language models predict a probability distribution over *the next word in an utterance given* n *preceding words*. More formally, if the utterance is represented as a sequence of words $\boldsymbol{W} = \langle w_0, w_1, \ldots, w_L \rangle$, a subset of this utterance $\langle w_i, w_{i+1}, w_{i+2}, \ldots, w_{i+n} \rangle$ is called an *n-gram*, and for brevity this will be written as $\boldsymbol{w_{i:i+n}}$. Given an n-gram $\boldsymbol{w_{i-n:i-1}}$, a language model should predict a probability distribution over a vocabulary of words such that each probability in the distribution describes the probability of the corresponding word being the next word (i.e., the $w_i{}^{\text{th}}$ word) in the sequence. For example, given an 3-gram input such as:

$$Today\ is\ the \tag{4.1}$$

one can expect a correctly functioning language model to put most of its probability mass into words like ordinal numbers (for example $first$,

so the full sentence could be something like *Today is the first day of the year*) or adjectives or nouns that one could relate to a day (for example $longest$, so the full sentence could be *Today is the longest day of the year*). Nouns and adjectives unrelated to days would presumably be less likely to follow. Parts of speech that would be ungrammatical after a determiner in a noun phrase (such as verbs or prepositions) are still less likely. In more formal terms one can write that a language model is trying to compute the following distribution:

$$\mathbf{P}(w_i|\boldsymbol{w}_{\boldsymbol{i-n:i-1}}) \tag{4.2}$$

Since the number of possible sentences in a natural language are presumed to be infinite, it is generally only possible to approximate a true language model. Doing so is a complex task, and machine learning techniques have long been used to construct useful language model approximations (Bahl et al., 1989). The idea of a *neural language model*, a language model approximation consisting of a neural network, has been a more recent focus in the field (Bengio et al., 2003; Salton et al., 2017). The basic concept is both simple and intuitive: a neural network is trained to learn to approximate the probability distribution presented in Equation 4.2. Although often slower to train than the traditional EM algorithm-trained n-gram models commonly used in the past[1], these mod-

[1] Although researchers are trying to improve this situation: see Mnih and Teh (2012) and Chen et al. (2015), for example

els perform extremely strongly (Kim et al., 2016), often representing the state of the art in the field.

4.2 Deep learning

Deep learning (LeCun et al., 2015; Kelleher, 2019) has been a major driver in recent progress in machine learning research. The term refers to the use of recent progress in both hardware and algorithms to train artificial neural networks that are much larger and more complex (in terms of having more layers, hence the term deep) than would have previously been possible.

Although it is likely that the reader has a degree of familiarity with the concepts and terminology of deep learning, a short outline of how an artificial neuron works will be provided here, both to refresh the readers memory, and to make the notation used in the thesis clear. A network is composed of a (usually very large) number of *computational units*, each of which can have any number of inputs, but will always have a single output, o. Suppose the unit has three inputs, denoted as i_1, i_2 and i_3 respectively. The output of the unit is then computed according to Equation 4.3 below:

$$o = a(i_1 w_1 + i_2 w_2 + i_3 w_3 + b) \tag{4.3}$$

Here, a is an activation function, usually something like a sigmoid or

hyperbolic tangent function, although more exotic choices are possible. Each input i_j is multiplied by a corresponding *weight* w_j, which may be positive or negative. The term b is called a *bias*, which becomes the primary determinant of the value of the output when the inputs are close to zero. Most actual implementations do not use an explicit bias, but rather have an extra input i_{n+1} (i_4 in this case) which has a fixed value of 1, and its associated weight w_4 acts as a bias. *Training* the network is achieved by running the *backpropagation* algorithm (Rumelhart et al., 1986) over a dataset and the network. In the standard neural network architecture, known as a feed-forward network, a network is composed of multiple *layers* of units, which are fully connected to all of the units in the immediately previous and succeeding layers, but not to any units in the same layer (although, as will be shown in Chapters 5 and 6, there are varieties of neural network architectures where there are recurrent loops in the connections in the network).

4.3 Using deep learning to extract structure from sequences

One of the earliest, and most successful examples of a deep learning model that can automatically learn structure is called an *autoencoder* (Hinton and Zemel, 1994). Autoencoders have proven very effective at compressing data into a shorter representation, typically called a *latent*

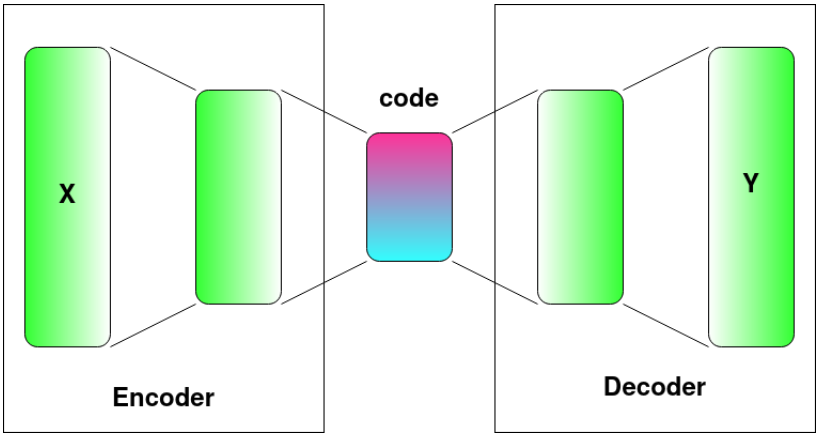


Figure 4.2
A typical autoencoder. Given an input X, the network attempts to reconstruct it at the output layer Y, learning a latent vector at $code$.

vector or *code*. Figure 4.2 represents a typical autoencoder setup. It consists of two parts: an *encoder* which consists of a number of layers of units, each one smaller (having fewer units) than its preceding layer, and a *decoder* which consists of the *same number* of layers of units as the encoder, but with the difference that each layer must be larger than its preceding layer. Actually, the final (output) layer of the encoder and the first (input) layer of the decoder are the same size, so the two networks are stuck together to form a structure that looks, depending on its orientation, much like an hourglass or a bow tie, as can be seen in Figure 4.2. In addition, the length of the encoder's input layer and the decoder's output layer must also be the same.

An autoencoder is trained to reproduce the input at the output layer after is has been passed through the information bottleneck of the smaller intermediate layers. More specifically, for each item x_i in the dataset, the network is trained in one piece, so that the input to the encoder and the

output from the decoder are both x_i. It is said that the network is learning to *reconstruct* x_i. In order for the decoder to carry the reconstruction correctly, the narrow layer in the middle must transmit important information needed for the reconstruction, but due to the space constraints it cannot transmit the entire input. Thus, the narrowest layer must contain information about the input sufficient to reconstruct it, but in a manner that is shorter than the input representation, usually by discarding unimportant information in a domain-specific manner.

When trained, the encoder can be disconnected from the rest of the network to allow compressed representations of inputs to be generated at will – the idea being that these compressed representations will encode just the most essential information necessary to reconstruct the input and will have filtered out extraneous information from the input.

Skip gram models have found wide use in the natural language processing community. They are similar to autoencoders, in that they produce an embedded vector output. Unlike an autoencoder, however, a skip-gram model learns to associate n inputs with a single output. Given a sentence $S = w_1, w_2, \ldots, w_L$, and for any $i \in \{1, 2, \ldots, L\}$, a skip-gram network receives $w_{i-1}, w_{i-2}, \ldots, w_{i-n}$ and $w_{i+1}, w_{i+2}, \ldots, w_{i+n}$ (i.e. n words before and n words after a particular word in the sentence) as inputs, and attempts to predict w_i (the particular word between the inputs) as output. Although skip-gram models and autoencoders are different in structure and the learning task they undertake, they are often deployed

with the same end-goal in mind, namely to learn a representation of the data. In the case of an autoencoder this is a compressed representation of the input sequence, and in the case of the skip-gram model this is a representation of the words in the vocabulary of the dataset that the input sequences are sampled from.

Since each word is being reconstructed by surrounding words rather than itself, skip-grams generally produce word embeddings that capture semantic information. A number of quite impressive examples of this have been observed in the literature. For example, Mikolov et al. (2013) published the image shown as Figure 4.3. This figure shows the relative positioning of words after the higher-dimensional vector representation for the words generated by a skip-gram model have been projected into a 2D space to for visualisation. The fact that the relative positioning of the words makes semantic sense to humans when they view the image is taken as evidence that the skip-gram model is learning some of the semantic relationships between the words. Remarkably, one can see that the translation between the embedding of a country and its capital city remains roughly constant.

This demonstrates that neural network models can learn complex semantic information about the domain that they are working in with relatively little data – the network trained by Mikolov et al. (2013) did not have any explicit knowledge of the concept of a country or a capital city, but appears to have learned the relationship between these two concepts

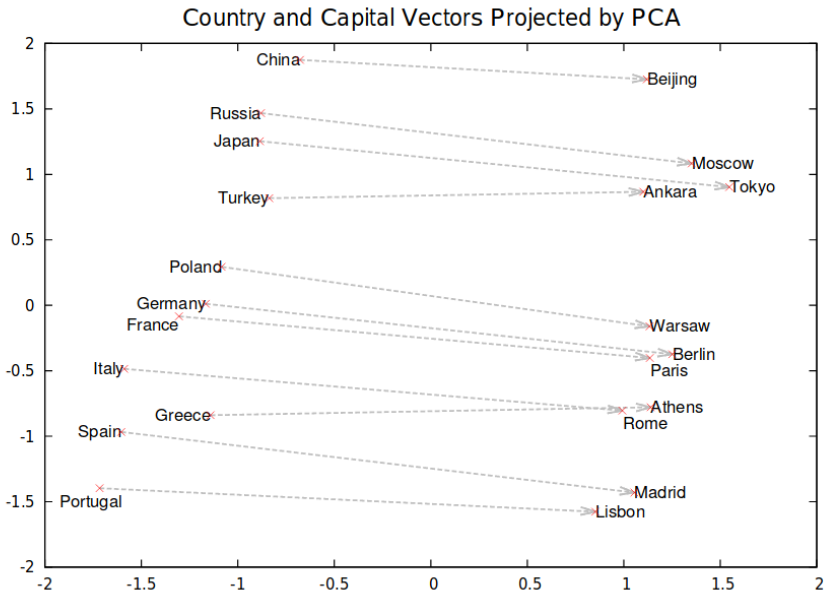


Figure 4.3
Word embeddings of countries and capital cities have remarkably similar translations. Taken from (Mikolov et al., 2013)

by itself, inferring that a relationship exists between them simply by observing the statistics seen in the dataset.

4.4 Activity discovery by means of a language model

At this point, a broad overview of the proposed approach can be given. The system documented in this chapter can be understood as being composed of two major components: a neural language model to compute probability distributions over *future events*, and another software component to convert these probabilities into links, much like the ones depicted in Figure 4.1.

The operation of this system proceeds in four steps. First, the neural language model is trained over the dataset, to allow it to predict future

events given a number of past events. Second, the model is used to build links between events. The fundamental signal for the creation of a link between events A and B is if when a language model is given an input context that includes event A, the language model predicts that another event B will happen in the future. For any particular activity discovery system the actual instantiation of a link between the events is controlled by a decision process working over the language model predictions. For now, however, the key insight is that the language model's predictions provide a basis for such linkage creation decisions. A group of related links are hypothesised to correspond to an activity. The relationship is treated as a *transitive* one: if events B and E are linked together, and C and E are linked together, the discovered activity then consists of B, C and E. Note that since D is not part of this activity, interleaving is accounted for. In fact, D could be part of some other activity, thus allowing the system to explicitly *disentangle* interleaved activities.

All of the activities found in the second step will be *instances* of an activity. An instance is a concrete sequence of sensor events forming an activity in a dataset. By contrast, multiple instances can share the same *type*, which means they represent the same sort of activity. This basic distinction between activity instances and types will be a recurring theme in this thesis. Because the output from the second step is activity instances rather than activity types, the third step is to *cluster* the discovered activities into activity types. The clustering method used is quite

simple: two activity instances are regarded as being of the same type if a majority of the events in one instance also occur in the other instance. Once this is done, the fourth step involves removing the events contained in each activity from the dataset and replacing them by a single new event, corresponding to its activity type. The process is then repeated with a new language model being trained on the updated dataset, and its predictions being used to create linkages between events in this updated dataset. This is the mechanism used to construct a hierarchy of activities, much like the one that was constructed for the topic modelling system in Section 3.3.

To construct a deep hierarchy of activities, these four steps are repeated many times, once for each layer of the hierarchy. The result of this process is a complex, multi-layered structure of disentangled activities.

4.5 Generalising language models

One issue with the approach presented in Section 4.4 is that neural languages models are typically designed to predict the *next* word/event given an input n-gram, but the model needs to potentially predict events that occur *more than one event in the future*. As a result, it is proposed that existing neural language models be modified to be able to handle this task. This section will introduce this *generalised neural language*

model.

Recall from Section 4.1 the concept of an n-gram, a partial utterance written $\boldsymbol{w}_{i-n:i-1}$. Mathematically, an n-gram is a form of *n-tuple*, a list of n elements that can be duplicated, and whose order is expected to be preserved. This contrasts with an *m-set*, a set of m elements, where each element must be unique and order is unimportant. If an n-gram is equivalent to an n-tuple, one can imagine an *m-window*, or *lookahead window*, which is a set of words equivalent to an m-set. One can imagine a function, f, which takes an n-gram as input and returns an equivalent m-window. Taking the n-gram $\langle longest, day, of, the, year \rangle$ as an example, f computes the following m-window:

$$f(\langle longest, day, of, the, year \rangle) = \{year, longest, the, of, day\} \tag{4.4}$$

f's input must be a sequence of words enclosed in triangular braces, which is because it is a tuple (an n-gram in this case). By contrast the output is a set (an m-window), and is thus enclosed in curly braces. In the above example, the content of the input tuple and the output set match, but this may not be the case. All of the usual restrictions applied to mathematical sets apply to an m-window. This means, for instance, that its elements can be re-arranged at will, and it must not contain duplicates. An example of this is shown in Equation 4.5 below:

$$f(\langle longest, day, day, of, the, year\rangle) = \{longest, day, of, the, year\} \tag{4.5}$$

The word *day* appears twice in the input n-gram, but should only appear once in the output m-window. The m-window in both Equations 4.4 and 4.5 are equivalent, since they consist of the same elements, even though their order is different, and they were produced from distinct n-grams. Since the input is of length 6 it can be referred to as a *6-gram*. In a slight abuse of terminology, the resulting m-window is not referred to as a *5-window* (since it contains 5 elements), but a *6-window* (since it was constructed from a 6-tuple). In order to keep the notation as clear as possible, $\boldsymbol{w}_{i-n:i-1}$ will continue to be used to refer *only* to an n-gram, and use $f(\boldsymbol{w}_{i-n:i-1})$ to refer to its equivalent m-window.

With the background now covered, the proposed generalisation of a language model can now be outlined. Given an n-gram $\boldsymbol{w}_{i-n:i-1}$, the network is trained to compute *a probability distribution over the contents of the following m-window*. This means that given an n-gram of n words, it tries to predict which words/events from the vocabulary are likely to appear within the next m words after the n-gram. Thus, a correctly trained language model of this type *should assign most of the probability mass of its output to the contents of the m-window*. This contrasts with a traditional neural language model in that it assigns a probability distribu-

tion not over the next word w_i, but rather over $f(\boldsymbol{w}_{i:i+m})$, i.e. over the m-window following the n-gram. This model's equivalent of Equation 4.2 would thus be:

$$P(f(\boldsymbol{w}_{i:i+m})|\boldsymbol{w}_{i-n:i-1}) \tag{4.6}$$

A typical language model is therefore a special case of this model, in which $m = 1$ (i.e. which is only trying to predict the next word). The distribution presented in Equation 4.6 is approximated by training a deep neural network, much in the way that traditional language models can be approximated by deep neural networks.

It is hypothesised that restructuring the standard language model definition in this way will be extremely useful for the purpose of activity discovery, since it can be used to construct dependency links such as the ones proposed at the start of this chapter.

4.6 Detailed approach

The actual network that was used to compute the probability distribution in Equation 4.6 follows the architecture shown in Figure 4.4, which is a variant of the network proposed by Bengio et al. (2003). For the experiments presented in this chapter, n-gram and m-window lengths were used with a value of 40, in other words $m = n = 40$. Figure 4.4 shows the dimensions and activation function of each layer. This section

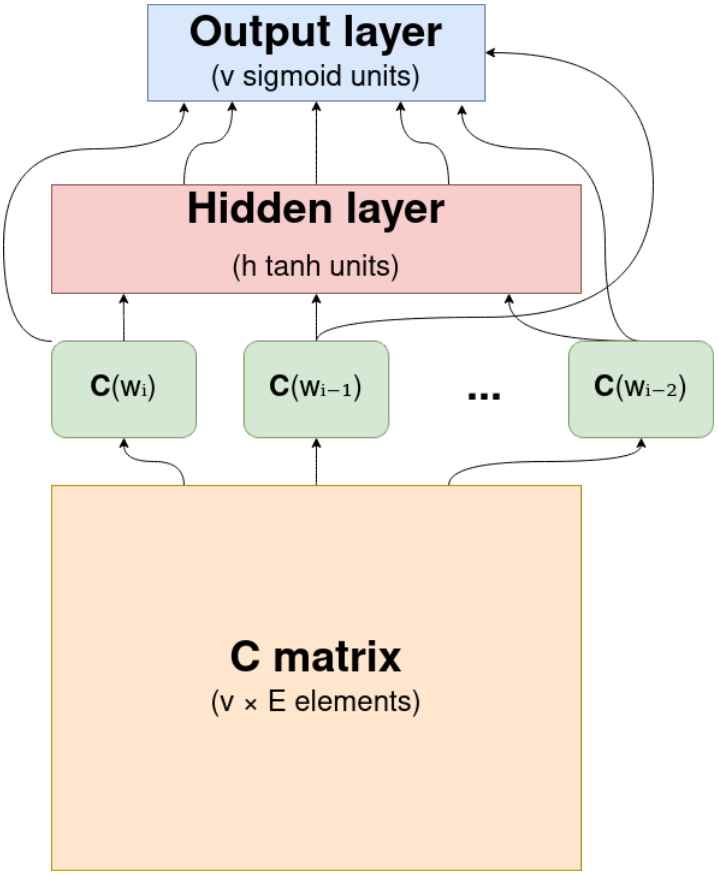


Figure 4.4
An illustration of the network architecture used in this chapter. The n-gram is fed in the bottom, and a probability distribution over m-window elements is produced at the top.

will now describe the network.

As with most neural language models, the input to the network consists of a series of n one-hot vectors over the entire vocabulary length v. This results in a very large ($n \times v$ elements) input vector. In order to reduce the computational complexity required to train such a large network, the one-hot encoded vectors are reduced to a fixed latent size E, as proposed by Bengio et al. (2003) (in this case $E = 100$). This is done by having a $v \times E$ matrix C. For each input word $\boldsymbol{w_i}, \boldsymbol{w_{i-1}}, \ldots, \boldsymbol{w_{i-n}}$ the one-hot vector encoding the word is replaced with the corresponding

row of the matrix. The matrix therefore has a row for each item in the vocabulary, and the row is of the length E, resulting in a $v \times E$ matrix. In Figure 4.4, one can see that the C matrix is used multiple times to encode the latent vectors for all of the inputs to the network, which are denoted $\boldsymbol{C}\left(\boldsymbol{w}_i\right), \boldsymbol{C}\left(\boldsymbol{w}_{i-1}\right), \ldots, \boldsymbol{C}\left(\boldsymbol{w}_{i-n}\right)$. These latent vectors then take *two* paths through the network. The bulk of the processing in the network is carried out by a hyperbolic tangent (tanh) *hidden layer*, with h units. For the network used in this chapter, $h = n \times E \times 1.5 = 6000$ units, which appears sufficient for the task at hand. The output from the hidden layer is then fed into the output layer to compute the final output. Remaining faithful to Bengio et al. (2003)'s original proposal, the output layer also receives the latent vectors ($\boldsymbol{C}\left(\boldsymbol{w}_i\right)$, $\boldsymbol{C}\left(\boldsymbol{w}_{i-1}\right)$, …, $\boldsymbol{C}\left(\boldsymbol{w}_{i-n}\right)$) as input. Thus the final output from each unit in the output layer is computed according to equation 4.7.

$$y = sigmoid(b + xW + tanh(d + xH)U) \tag{4.7}$$

Here, x denotes the concatenated latent vector (i.e. the vector resulting from the concatenation of $\boldsymbol{C}\left(\boldsymbol{w}_i\right)$, $\boldsymbol{C}\left(\boldsymbol{w}_{i-1}\right)$, …, $\boldsymbol{C}\left(\boldsymbol{w}_{i-n}\right)$ horizontally), and so is the input to the hidden layer. The weights and biases for the hidden layer are denoted H and d respectively. Because it is taking two separate inputs, x and the output of the hidden layer, the output layer has two weight matrices, U for the input from the hidden layer and

W for the input vector x. The biases of the output layer are denoted by b.

Recall that the network has to predict the next m events to take place, so the number of outputs could be any integer in the range $[1, m]$. For this reason, this network uses *sigmoid activation function*, in contrast to the softmax function used by Bengio et al. (2003). Softmax functions are usually only appropriate in situations where a single output is needed. The network was trained using generic backpropagation. The optimiser used was vanilla gradient descent, with an exponentially decaying learning rate. One issue that was found when training the network is that the "usual" initial value ranges for the network (typically, values between -1 and 1) fail to converge at training time. When initialised to very small values, however, the network does converge. Through a process of trial and error, it was found that values drawn uniformly in the range -0.01 and 0.01 resulted in the network successfully minimising its loss function during training faster than other initial values. This initialisation procedure worked for both of the datasets the approach documented in this chapter was tested on. These two datasets (SCARE and Opportunity, as discussed in Section 2.8) are similar to many other datasets used in the field of activity discovery, although this initialisation procedure may nonetheless need to be revised in order to use some datasets.

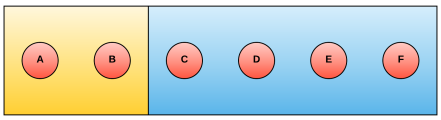


Figure 4.5
A stream of sensor events with an n-gram sliding window over previous events to the left and an m-window of future events to the right highlighted

4.6.1 Building links

Of course, the network is just one component of the complete system. As mentioned in Section 4.4 the proposed system is in fact a hybrid system, combining a neural network trained using backpropagation with a manually engineered system, which uses the probability distributions computed by the network to find activities in the dataset. Figure 4.5 represents (an extract of) the input dataset, where each circle labelled with a letter represents a single sensor event. The yellow/light coloured rectangle on the left covering events A and B represents an n-gram (2-gram in this case) over previously observed events. The blue/dark coloured rectangle on the right covering the remaining events will be provided as input to the function f to compute a 4-window over future events. The network's task is to predict what event types are likely to lie within the dark coloured rectangle given the contents of the light coloured rectangle as input. This rectangle of events is called the *m-future-gram*, to distinguish it from both the n-gram and the m-window (which is computed from the m-future-gram).

It is extremely important to note that the probability of any particular

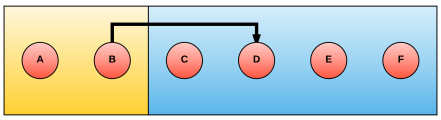


Figure 4.6
A link connecting events B and D, forming an hypothesised activity

event type will fluctuate depending on the elements in the input 2-gram. Suppose the probability of event D (or more precisely, the probability of an event of whatever type D is) increases when the n-gram contains event B compared to when it does not. This can be taken as an indication that events B and D are part of the same activity, since knowing that B has occurred allows one to predict that D will follow shortly after. In that case, the system builds a *link* between events B and D, as shown in Figure 4.6. Thus, the system doesn't actually use the raw probabilities output by the language models, but rather the *differences between* probability vectors where B is present and vectors where B is not present. In this thesis, these values are called *probability deltas*.

Actually, the above description is a slight simplification of the full procedure. Some sensor events will likely be more common than others, and if these keep being predicted as likely to occur by the model it could lead to the discovery of spurious links. Therefore, for each n-gram, the network actually takes $n+2$ *sub-n-grams* as input. In the example shown in the diagram, the input consists of the event preceding A as well as A itself, followed by A and B, followed by B and C (as depicted by the outline in Figure 4.8), and finally followed by C and D. Rather than

analysing the raw probabilities, the system then computes the deltas over the probability of observing event D in the sub-n-grams containing B as compared to the sub-n-grams not containing B. This approximates the conditional probability of D given that it is know that B is present, allowing a link to be built between the two events if *this* probability exceeds some threshold. The threshold could either be supplied by the user, or could be computed at runtime based on the probabilities observed (both approaches were tried, and both provided similar performance).

Using the example illustrated in Figure 4.5, rather than directly using the raw probability distribution computed by the language model when A and B are given as inputs, this approach instead averages the distribution when B is present in the input (so for input $\{A, B\}$, but also for the next sliding window input, $\{B, C\}$), and subtract it by the average distribution for the input $\{C, D\}$ and $\{event\_before\_A, A\}$. The exact calculation used is presented in Equation 4.8, where P_{i+n} denotes the probability vector produced by the language model when given the n-gram sliding window $D_{i+n:i+n+m}$ as input.

$$delta(P_{i+n}) = \frac{\sum_{k=i+n}^{i+n+n-1} P^k}{n} - \frac{P^i + P^{i+n+n}}{2} \tag{4.8}$$

Pseudocode for this process is presented below as Algorithm 7.

A link is built between d_{i+n} and d_{i+n+l} *if and only if this probability delta exceeds a certain threshold.* The thresholds are computed at

```
def compute_delta(P, i, n):
    x = vector of zeros
    for k in range(i + n, i + n + n - 1):
        x = vector_add(x, P_k)
    x = vector_divide(x, n)

    y = vector_add(P_i, P_{i+n+n})
    y = vector_divide(y, 2)

    return vector_subtract(x, y)
```

Algorithm 7: Computing probability deltas

runtime, since different event types need different associated thresholds for the link-building to work correctly. A number of automatic discretisation algorithms were tested. One commonly used method to compute a binary threshold is Otsu's method (Otsu, 1979), which is commonly used the image processing community. This works by converting an image to greyscale, and then producing a histogram over its pixel intensities. The threshold is the point in the histrogram where the integral of pixel densities to both the left and right of the threshold are equal. This is mathematically equivalent to computing the k-means clustering with $k = 2$ for the pixel intensities and then taking the average of the two centroids as a threshold (Liu and Yu, 2009). However, this method assumes that the histogram in question has two distinct *humps*, one for light pixels and another for dark pixels. This is unlikely to be the case for the datasets used in this thesis. In this case, thresholds can be computed automatically by applying Otsu's method, dividing the image into bright and dark sub-images based on the threshold, running Otsu's method over

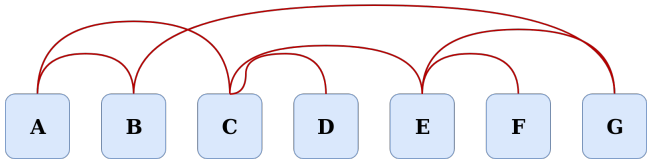


Figure 4.7
Links are built between events if an LSTMs probability delta passes a threshold. This threshold is computed dynamically at runtime.

both sub-images, and then using the average of these two thresholds as the threshold for the next iteration. This iterative process stops when the threshold begins to converge, i.e. when the threshold computed in two iterations falls below a certain epsilon. This is the method used to automatically compute thresholds per event type, using probability deltas in place of pixel intensity values. The linking process results in a tangle of binary links between events, as illustrated in Figure 4.7.

Assuming that the thresholds are placed in a dictionary or map-like data structure called thresholds (so that thresholds[evnt] denotes the threshold required to build a link to events of type evnt), then the function build_link (presented as Algorithm 8) will return true if a link is to be built between events d_{i+n} and d_{i+n+l}.

```
def build_link(P, d, i, n, l, thresholds):
    delta = compute_deltas(P, i, n)
    expected_event = d_{i+n+l}
    if delta[index_of(expected_event)] ≥ 0 and \
       delta[index_of(expected_event)] ≥ \
       thresholds[expected_event]:
        return True
    return False
```

Algorithm 8: Function to determine if a link should be built between events d_{i+n} and d_{i+n+l}

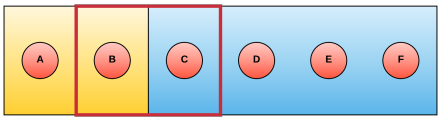


Figure 4.8
For each n-gram (like that depicted in Figure 4.5), $n + 2$ sub-n-grams are fed into the network to compute the conditional probability of D given B. The sub-n-gram consisting to B and C is shown here

The links are then grouped together to form activities. For example, if a link is built between the sensor events at indexes 1 and 5, and another link is built between 5 and 8, the system will output an activity consisting of 1, 5 and 8.

The n-gram and m-future-gram are then incremented by one event and run the procedure again. This may build another link between C and some other event, or may not result in any link. Links can also cross other links, so it is possible to build complex representations of interleaved activities.

Once links are built, they are clustered into *activity types* based on similarity. As noted in Section 4.4, a similarity threshold has to be selected for this – the threshold used here was 60%. Thus the output consists of a set of activities and a type associated with each activity.

4.7 Results

The evaluation was started by testing the system on the SCARE data-set, which allows its performance to be directly compared to the topic modelling system presented in the previous chapter. The basic means of

Table 4.1
Performance of the feedforward language model system on the SCARE dataset, with n-gram and m-window lengths of 40, with a hierarchy of depth 4

Activity	Label ID	Precision	Recall	F1 score
Move Quad	6	0.7755	0.16	0.2653
Move Picture	2	0.6019	0.1809	0.2782
Move Rebreather	3	0.4963	0.0217	0.0416
Move Silencer	0	0.5291	0.2717	0.359
Move Box	8	0.1875	0.07021	0.1022
Macro average		0.5181	0.1409	0.2093

evaluation was the same. The results are shown in Table 4.1.

The results show that this system slightly trails behind the performance of the LDA-based system tested in Chapter 3. In particular, this system has a macro averaged F1 score of 0.209, compared to 0.36 for the LDA system. The micro averages are 0.219 for this system and 0.486 for the LDA system. Nonetheless, it is likely that the gap between the two systems can be closed, which is something that will be covered in later chapters.

The system was then evaluated on the Opportunity dataset. The data was preprocessed into a stream of discrete events using the same basic procedure used for the SCARE dataset (see Section 3.4 and Section 2.8). The basic means of evaluation was also the same. The results are shown in Table 4.2. The “Activity" column contains activities that exist in the dataset’s ground truth, while the “Label ID" column shows the discovered activity that was found to correspond most closely to each ground truth label. The correspondences were computed using the same

Table 4.2
Performance of the feedforward language model system on the Opportunity dataset, again with inputs of length 40 and a 4-layer deep hierarchy

Activity	Label ID	Precision	Recall	F1 score
Stand	2	0.66	0.085	0.15
Walk	7	0.636	0.109	0.186
Sit	11	0.6	0.104	0.177
Lie	0	0.629	0.096	0.167
Other	1	0.535	0.110	0.182
Macro average		0.612	0.1008	0.1724

Figure 4.9
An extract from a visualisation of the neural language modelling based system against the Opportunity dataset

ground matching algorithm documented in the previous chapter (see Algorithm 6 on page 88).

Examining the results of this generalised language model system on both the SCARE and Opportunity datasets, it is striking the that system has much lower recall than precision. To determine why this is, a visualisation of the system's output was produced, which is shown as Figure 4.9.

The red bars at the top represent ground truth activities, while the blue bars at the bottom represent discovered activities. The bars are arranged in rows, each of which represents a single *activity type*. There are two striking aspects to this diagram: one is that the number of activities

output by the system is far larger than that which exist in the ground truth. The second is that the output activities tend to be quite *fragmented* (i.e., one can see a lot of instances of something called *fragmentation error*, a concept taken from the Ward et al. (2006) paper, which will be presented in Section 7.2). From the point of view of the evaluation metrics being used, this results in a large number of *false negatives* being observed, which explains the low recall (which punishes false negatives). However, two things should be taken into account when considering this result:

- Real activities tend to be noisy. Not all sensor readings observed during a *making dinner* activity will be related to making dinner. If the system excludes these events from its output, it is actually producing output more correct and detailed than the ground truth, but is actually being punished for it

- Since this works puts an emphasis on hierarchies, it would be useful to be aware that the output from the system could be *on a different level of abstraction* to the dataset. The outputs may not be activities, but sub-activities. In other words, the system may not find *making dinner*, but it may have found *chopping vegetables*, *serving* and *preparing gravy* as separate activities.

This leads to two points: firstly, one should focus on trying to improve the *precision* of the system, rather than the recall or overall F-measure,

since that is the best way of ensuring that it finds something that corresponds to activities present in the dataset's ground truth, while avoiding mis-evaluating systems. Secondly, this means the neural language model-based system presented here is performing better than the raw F-measure seems to suggest.

For the remainder of this thesis, therefore, improving the precision of the systems presented will be the primary goal.

4.8 Summary

In this chapter, a novel approach to activity discovery has been presented and evaluated. It is worth taking some time to analyse the results presented to help determine the viability of the approach.

As noted in Section 4.7, the precision of the system is consistently higher than the recall. This suggests that the system rarely returns false positives (i.e. it rarely adds an event to an activity that is not actually part of the activity). The low recall suggests that the system *does* frequently return false negatives (i.e. it fails to add events to an activity that are part of that activity). This suggests that the system in quite conservative, and only adds events to an activity/link if it is sure that the event belongs there.

Another interpretation is that the activities found by the system are consistently *subsets* (or sub-activities) of the true activities present in the

dataset. This leads to the argument that the system is finding activities at a different level of abstraction to the ground truth – it finds activities present in the dataset, but it disagrees with the ground truth annotation of where those activities begin and end, and what specific events constitute the activity. This is actually a good thing, since the hierarchical nature of the presented approach can deal well with activities that are small subsets of larger, more complex activities. It also suggests that the system is disentangling interleaved activities, although more work will have to be done in later chapters to confirm this.

Finally, this chapter has also given a stark example of the complexities of fairly evaluating activity discovery systems. Comparing the output of an activity discovery system to a ground truth can often be a poor means of evaluation, which obviously isn't the case in other fields of machine learning. This thesis will return to discussing this issue in Chapter 7.

Chapter 5

Overcoming Interleaving with Recurrent Modelling

The last chapter made use of a language model first proposed by Bengio et al. (2003). This is a quite an old neural language model, and is one that has long been superseded in the literature. In particular, models of this type seem to perform poorly at modeling long-range dependencies, which is a major problem for for modelling events in an activity dataset, since it is presumed long-range dependency modeling is an essential part of identifying interleaved data.

Most modern neural language models attempt to resolve these problems through the use of a *recurrent* neural architecture. A recurrent network differs from non-recurrent designs by supplementing the connections between layers found in all networks with connections *through time* (Kamijo and Tanigawa, 1990; Zaremba et al., 2014). In other words, the network layers receive additional inputs produced by themselves *at*

a previous timestep. Recurrent language models have already been used successfully in the NLP community to deal with long-range dependencies in language (Mikolov and Zweig, 2012; Mahalunkar and Kelleher, 2018a), which provides a good case for trying them for event prediction. As will be seen later, the results show that recurrent networks exhibit high performance on the datasets.

This chapter also presents a number of major improvements to the link building procedure outlined in Section 4.6.1. Initially, a new algorithm was developed for building non-transitive links between activities, because of the unsatisfying performance of the threshold based procedure used in the last chapter. The new algorithm is detailed in Section 5.3.

This chapter will detail the specific network architecture that was used (Section 5.1) and how the architecture's performance was validated (Section 5.2). The extensive changes made to the link-building process are then detailed (Section 5.3), followed by the experiments that were conducted on the new system (Section 5.4). Finally, results are presented in Section 5.5.

5.1 Language modeling with Long short-term memory

The new network architecture is a recurrent, LSTM-based, design. *Long short-term memory* (LSTM) networks are a form of recurrent network that were first proposed by Hochreiter and Schmidhuber (1997). They

address the *vanishing* (or *exploding*) *gradient* problem, where a recurrent network operating over a long period of time has difficulty backpropagating the error gradients back through the network to update the weights in the early layers of the network as the span of time the network covers increases. Note, that in a recurrent architecture the network is as deep (in terms of the number of layers there are in the network) as the number of time-steps (or inputs there are in the sequence) it is processing. There are a number of factors that contribute to the phenomenon of vanishing gradients in recurrent architectures but the primary cause of this difficulty is that the backpropagation of error gradients back through a network involves the error gradients being repeatedly multiplied by the weights on the recurrent links in the network. This multiplication of the error gradients by the weight on the recurrent links occurs once for each layer in the network (i.e., once for each time-step the error gradients are backpropagated through). However, these weights are generally smaller than 1 and so these repeated multiplications result in the error gradients getting smaller and smaller each time they are propagated back through a layer of neurons. Eventually the error gradients become so small that they no longer contain a useful signal for the adjustment of the weights, in other words the gradients have vanished, and this results in the training of the weights in the early layers of the network becoming very slow.

An LSTM is a recurrent network with an attached memory storage unit called a *cell*. The cell stores a single unit of information, potentially

over a very long period of time, that the network can access as needed. The contents of the cell, and the degree to which its contents influence the output, is controlled by a series of *gates*, which operate much like a feedforward unit. A typical LSTM might have an *input gate* (which controls the degree to which the LSTMs input at a particular step influences the cell's value), a *forget gate* (which controls the degree to which the LSTM cell's value in the previous iteration controls the current value) and an *output cell* (which controls the degree to which the LSTMs output is influenced by the cell's value).

This work specifically uses the LSTM-based language model presented by Zaremba et al. (2014) (which is itself adapted from Graves (2013)), which allows for the modelling of long-term dependencies and robustness to noise. The LSTM consists of an input gate i, a forget gate f, an input modulation gate g, and an output gate o. If h_t^l denotes the output of layer l at timestep t, and c_t^l denotes the value of an LSTM cell in layer l at time t, activation of the gates is defined as:

$$\begin{pmatrix} \mathrm{i} \\ \mathrm{f} \\ \mathrm{o} \\ \mathrm{g} \end{pmatrix} = \begin{pmatrix} \mathrm{sigm} \\ \mathrm{sigm} \\ \mathrm{sigm} \\ \mathrm{tanh} \end{pmatrix} W \begin{pmatrix} \mathrm{h}_t^{l-1} \\ \mathrm{h}_{t-1}^{l} \end{pmatrix} + b \tag{5.1}$$

The value in the cell is updated according to the following equation:

$$c_t^l = f \odot c_{t-1}^t + i \odot g \tag{5.2}$$

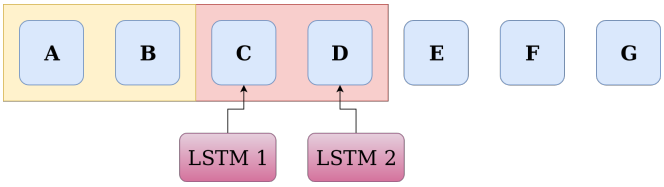


Figure 5.1
In this example, events A and B are inside the sliding window, and are fed as input into two LSTMs. Each LSTM has to predict a probability distribution over the corresponding offset within the lookahead window containing C and D.

where $\odot$ denotes the *Hadamard product* (i.e. elementwise multiplication, as opposed to matrix multiplication). The output h_t^l is then:

$$h_t^l = o \odot tanh(c_t^l) \tag{5.3}$$

Unlike the big, single network of the previous chapter, m networks are trained in this case, one for each lookahead offset. As a result, one network is responsible for predicting the first item in the lookahead window, another for predicting the second and so on. This process is illustrated in Figure 5.1.

5.2 Validating the LSTM's performance

Before proceeding further, it would be useful to verify that the language model works correctly before going on to other parts of the system. Recall that for each position in the look-ahead window a separate LSTM model is trained to predict the next event at that position: LSTM 1 predicts the event that will occur one time-step into the future, LSTM 2 predicts the event that will occur two time-steps into the future, and so on. More specifically, for a given input sequence each of these LSTMs

predicts a probability distribution over the set of possible events that describes the probability of each possible event being the next event at the relevant position in the look-ahead for the LSTM. In assessing the accuracy of each of these models this validation process judges the LSTM model to have been correct in its prediction if the actual event that is present in the data at the relevant position in the look-ahead appears as one of the top three predicted events in the probability distribution generated by the model.

This experiment used the Kyoto 3 dataset gathered by the CASAS project (Cook and Schmitter-Edgecombe, 2009), which was introduced previously in Section 2.8. Kyoto is a smaller, simpler dataset than Opportunity, but is still highly interleaved, and is based on real-world (as opposed to virtual) activities. In total, the preprocessed data contains about 7000 events of 100 types. Some of these events are very common (and occur over 500 times), while others are far less common (about 13 of the events occur less than 10 times in the entire dataset, for example).

An extract of the results achieved are shown as Figure 5.2. This shows that the LSTM language model is clearly able to successfully predict the contents of the dataset. It has an accuracy of 98% for the first item in the lookahead window (offset 1), but it drops down as it tries to predict further along the offset. This is precisely what would be expected, since predicting further into the future is an inherently harder problem. The accuracy is always high, it never drops below 94%.

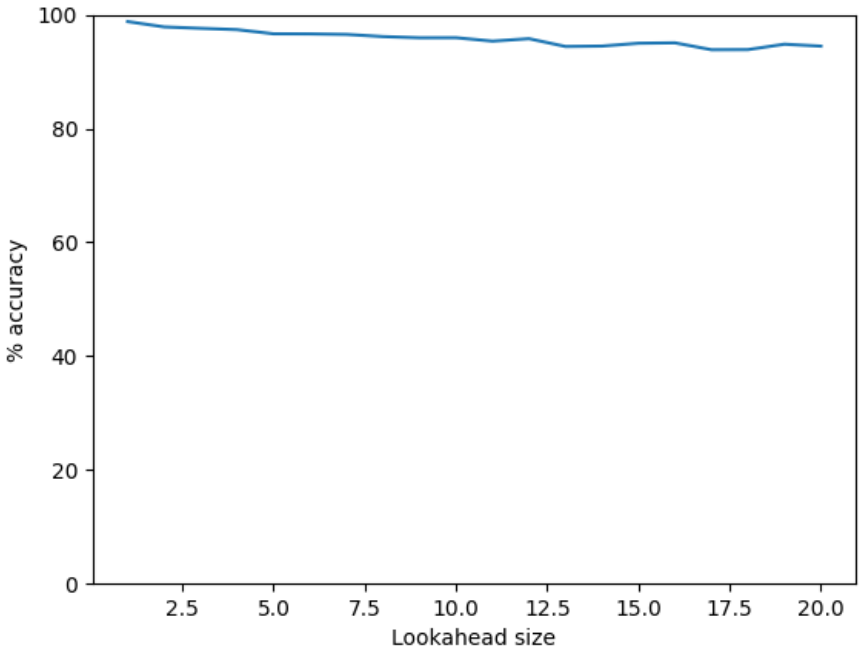


Figure 5.2
Average performance of the LSTM models on the Kyoto 3 dataset.

5.3 Building non-transitive links

The results in the previous section clearly indicate that this language model is able to predict future events from past events. When the model is dropped into the implementation described in Chapter 4, the initial result is that *every event* is linked together into a single activity. This necessitates finding a new way to build the links that avoids this problem.

The link-building process that has been used up until now has been transitive. That is to say, if the system builds a link between events A and B, and another link between events B and C, it places A, B and C into the same activity. This amounts to assuming an implicit link between A and C. An obvious first step for solving the excessive linking discussed

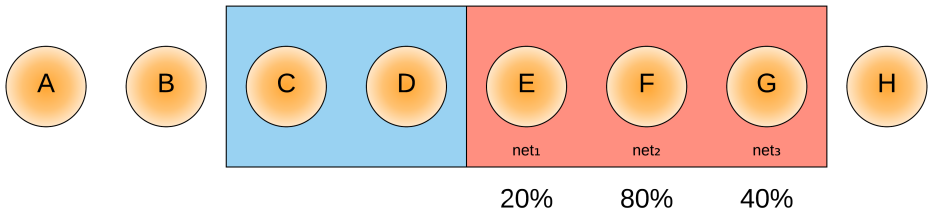


Figure 5.3
net_1 assigns a 20% probability to offset (j) 1 being equal to event type E, net_2 assigns an 80% probability of offset 2 being equal to event type F, and net_3 a 40% probability of offset 3 being equal to event type G. Note that the raw probabilities are converted to *probability deltas* before being used.

above is to simply remove the transitive linking, and only allow binary links. This is obviously not a solution to the problem but rather a first step in trying to find a more principled means of linking.

The new (binary) linking system which was developed has previously been published as (Rogers et al., 2020a). As before, the procedure assumes a sliding window of length n which iterates over a the entire dataset, followed by a lookahead window of length m. Given a particular position, consisting of a sliding window index i, and a lookahead offset j, the linking process begins by iterating over each $i \in \langle 1, 2, \ldots, n \rangle$ and each $j \in \langle 1, 2, \ldots, m \rangle$. For each (i, j) pair, one can view $\mathbf{P}_{ij}$ as the probability delta distribution that each $d \in \Sigma$ will be the j^{th} event *after* the i^{th} event.

In the case of Figure 5.3, it can be seen that net_2 has assigned a high probability to the event type F occurring at an offset of two after D. By contrast, net_1 and net_3 have assigned much lower probabilities to their corresponding values. Thus, one would expect that this means that

events D and F are part of the activity, which would justify connecting them via a link.

This is essentially a greedy linking strategy as it guarantees that only the strongest links for a given symbol are created. So far, this is very similar to the linking algorithm described in Chapter 4. Recall that in Section 5.2, it was shown that the LSTMs at larger offsets into the sliding window have a harder time making accurate predictions. This makes intuitive sense, since predicting the *next* event will generally be easier than predicting the event that will occur three events from now. Analogously, predicting the next word in a sentence is easier than predicting the word that will come a number of words after. This provides justification for modifying the algorithm to explicitly take distance into account, so longer offset networks get a small boost in their probabilities to offset the inherent higher difficulties in what is expected of them.

Thus, each probability delta is multiplied by a *correcting factor* that is equal to 1 for an offset of 1, some value larger than one for offsets greater than 1, and which increases linearly. The parameter which controls the degree with which the factor increases is called x. Since this value is no longer a valid probability it instead referred to as a *score*. Thus, the formula for computing the score for offset $\mathbf{P}_{ij}$ is as follows:

$$score(\mathbf{P}_{ij}) = \mathbf{P}_{ij} \times (1 + \frac{j}{x}) \tag{5.4}$$

This link will *only* be built if D and F do not link more strongly with some other event in the dataset. For example, if D was predicted with a probability delta of 90% when the sliding window ended at event B, a link would be built between B and D instead.

After building the links, they are then pruned by applying a threshold factor, which is called y, to remove spurious links. Link types that do not appear at least y times in the entire dataset are removed.

5.3.1 Clustering activity instances into activity types

The next step is to match all the links that were found with links of the same type. This step is called *clustering*. In the case where a link is built between F and D, one would need to cluster all other links between event types D and F (or equivalently between F and D) together. Note that this differs from clustering in the usual sense of the word, since the system is trying to find exact matches between link types, not semantic similarity as would be done in a clustering algorithm such as k-means clustering.

5.3.2 Building a hierarchy

The final step in a single iteration of the model is to build a new dataset, where each discovered link of two activities is removed, and replaced with a *new event*, with each activity type giving rise to a new event type. The outcome of this process applied to the small dataset that has

Figure 5.4
Events D and F have been removed and replaced with the new event $New\ event$. One can train a new set of LSTMs for this dataset, allowing hierarchies of activities to be discovered.

been used as an example in this section is shown as Figure 5.4. This allows a new set of m LSTM networks to be trained, after which the entire process described in this section can be repeated again. At the end, a tree-like structure will result, showing a hierarchy of (possibly overlapping) activities contained within each other. This is inspired from the way the Sequitur algorithm (Nevill-Manning and Witten, 1997) (previously mentioned in Section 2.4) constructs tree structures from sub-sequences that occur multiple times in a sequence, generalised to allow for non-contiguous sub-sequences. Ideally, the process would be run until a sufficiently high level of abstraction (where the tree-like structures correspond to activities) has been reached. In practice, the process can be stopped early if only a partial result is needed. The new event could be placed into the position formerly occupied by either event D or event F. The choice shouldn't affect the evaluation metrics being used, so the choice of which position is somewhat arbitrary. In this case, the new event is placed in D's position.

5.4 Experiments

The approach described in Sections 5.1 and 5.3 was tested on both the SCARE dataset from previous chapters (Stoia et al., 2008), and the CASAS Kyoto 3 dataset described in Sections 5.2 and 2.8 (Cook and Schmitter-Edgecombe, 2009).

As noted in Section 5.1, the system is an implementation of the LSTM model proposed by Zaremba et al. (2014). The implementation is written in the Python programming language, using the TensorFlow library (Abadi et al., 2015) running on an Nvidia graphics card using CUDA. The model was trained using the ADAM optimiser (Kingma and Ba, 2014). The hierarchy was trained for 5 layers: each layer took roughly an hour to train and cluster. Each network consisted of a stack of two LSTM layers, with a width of 150 LSTM cells per layer. A sliding window length of $n = 20$ was used, a lookahead window length of $m = 10$, a score factor $x = 400$ and an event type threshold $y = 3$. As with the system described in Chapter 3, these meta-parameters were chosen by means of a grid search over a number of parameterisations – the values above were found to result in the highest performance of the system according to the evaluation metrics discussed in the next section (Section 5.5).

5.5 Results

This section presents the results of the experiment described in Section 5.3. For the reasons outlined in Sections 4.7 and 4.8, these results concentrate on the precision for this evaluation, since the recall and F1 values are likely to be misleading.

In order to allow comparisons with the previous systems, the system was first tested on the SCARE dataset. The system discovers in excess of 150 activity types, so reproducing the full result of this evaluation here would not be possible. Nonetheless, an extract from these results is presented as Table 5.1. In the interests of completeness this includes the recall and F1 scores for the system, but thos can be regarded as being secondary in importance to the precision because of how recall inherently disadvantages the system and gives an inaccurate view of its performance. The fact that the number of activity types being found is much larger than the number of activities in the ground truth indicates that – much like the system presented in Chapter 4 – this system is consistently finding activities at a much lower level of abstraction than those that exist in the ground truth. In order to compute the evaluation metrics, each discovered activity type is associated with a single ground truth activity type using the same algorithm presented for this task in Section 3.5 (see Algorithm 6 on page 88). This means that each ground truth activity can be associated with more than one discovered activity.

Again, it is expected that this will result in a system that scores well on the precision metric, but poorly on both the recall and F1 metrics.

The results are reasonably good: more than half of the events discovered correlate to a precision of at least 50%, meaning that the results show a correlation (but not a perfect overlap) between the ground truth and the discovered output. The total macro average precision (over the entire dataset) is about 70%, with the micro average being slightly higher at 72%. This compares favourably to the previous two systems – the macro average of the precision for the LDA and basic language model was about 60%, although the LDA did achieve a higher micro-average of about 80%. Considering both the differing levels of abstraction, and the interleaving present in the SCARE dataset, this appears to be an acceptable initial result. The large number of events suggests that in the future, more needs to be done to combine the discovered event types.

The system was also tested on the Kyoto 3 dataset – the results of this are presented as Table 5.2. The results are again quite good, with the macro average of the total precision being about 66%, and the micro average about 67%. The macro- and micro-averages for the F1 score are about 18% and 19% respectively. The performance is slightly higher for the SCARE dataset, but no changes were made to the system other than the choice of dataset, so this only proves that the system can better handle the SCARE data.

As mentioned earlier, another important evaluation metric is the com-

Event name	Precision	Recall	F1 score
	...		
new_event_33	0.4676	0.0721	0.1249
new_event_40	0.6667	0.3043	0.4179
new_event_82	0.6667	0.2971	0.4110
new_event_18	0.3333	0.1361	0.1932
new_event_81	0	0	0
new_event_100	0.6667	0.2629	0.3771
new_event_65	0.6667	0.0472	0.0882
new_event_75	0.8279	0.1051	0.1865
new_event_19	0.6667	0.1412	0.2330
new_event_9	0.6407	0.1583	0.2538
new_event_17	0.6667	0.1867	0.2917
new_event_101	0.7560	0.0715	0.1307
new_event_50	0.7261	0.3089	0.4335
new_event_87	0.7474	0.3322	0.4600
new_event_72	0.7410	0.2941	0.4210
new_event_52	0.5288	0.0959	0.1623
	...		
Macro average	0.6106	0.1758	0.2616

Table 5.1
Results obtained when using the SCARE dataset

pression rate. The LSTM system compresses the original input datasets to about 68% of the original input size. This is a good result, albeit one that can hopefully be improved upon in the future.

One issue that affects this system is that the hierarchy building quickly runs out of steam. The compression ratio of 68% reported above applies to the top layer of the output hierarchy. However, the compression ratio of the bottommost layer of the hierarchy is almost as good, at about 65%. Thus, although the system builds many links the first time it encounters the dataset, it fails to generalise the discovered activities as expected, and fewer new activities are found in subsequent layers.

Finally, a graphical visualisation of the system's output was produced,

Event name	Precision	Recall	F1 score
	...		
new_event_10	0.2857	0.0051	0.0101
new_event_11	0.6667	0.2677	0.382
new_event_12	0.6667	0.3016	0.4153
new_event_13	0.3333	0.0005	0.0009
new_event_14	0.4286	0.0186	0.0356
new_event_160	0.75	0.1849	0.2966
new_event_161	0.6667	0.1893	0.2949
new_event_162	0.3333	0.0407	0.0725
new_event_163	0.6667	0.0282	0.0542
new_event_231	0.6667	0.3205	0.4329
new_event_232	0.6667	0.1266	0.2128
new_event_233	0.6667	0.185	0.2897
new_event_301	0.7143	0.0606	0.1117
new_event_302	0.6667	0.2138	0.3238
new_event_303	0.3333	0.0926	0.145
new_event_304	0.25	0.0162	0.0304
new_event_305	0.3333	0.0385	0.069
	...		
Macro average	0.535	0.123	0.1869

Table 5.2
Results obtained when using the Kyoto dataset

Figure 5.5
A visualisation of the system output, where the red bars at the bottom correspond to ground truth activities, and the triangles correspond to discovered events that can be understood as *compressing* the original dataset. The hierarchy is shown to be very deep in places.

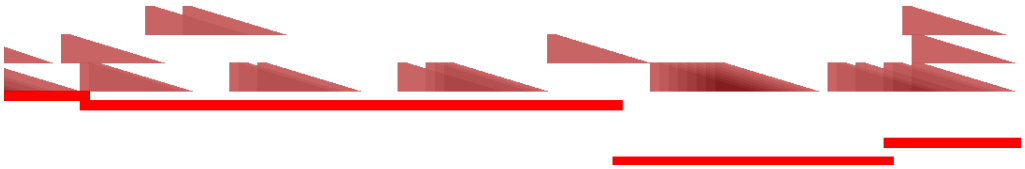


Figure 5.6
Although events sometimes cross activity boundaries, the incursions are always small, indicating they could still be part of the activity.

which is presented as Figure 5.5 and Figure 5.6. At the bottom of these images are red bars that represent activities present in the ground truth of the dataset. The bars are arranged in rows, and each row represents a single activity type. A bar is present when this activity is active, and absent when it is not active. Above these bars are right-angled trapezoidal or triangular shapes, representing the discovered activities output by the system. The wide bottom of each triangle represents the length of a group of events that was compressed into a shorter group by the system as it discovered activities and replaced the low-level events with these new activities. The length of the top of the triangle represents the length of the resulting shorter group of events.

In the specific case of Figure 5.5, one can see a large number of triangles have been found, and have formed a hierarchy which is 4 layers deep, visualising a run of the system over 4 layers. The fact that the triangles seem to cluster around the central activity in the image with relatively little overlap can be seen as indicating that the system has identified that the events taking place in that part of the dataset are significant, even it it hasn't found the entire activity.

Only a handful of the triangles are crossing activity boundaries in Figure 5.5, although it can be seen happening to a greater extent in some parts of the dataset, for example in Figure 5.6. Even here, however, the incursions are relatively small. This pattern repeats itself throughout the entire image: the ground truth and the discovered activities usually have an unusually high level of agreement. When the incursions do occur, they could be evidence that the human annotator of this dataset and the system are seeing similar activities, but cannot agree when they start or end as discussed above. This visualisation thus provides insights that other, more quantitative evaluation metrics generally fail to provide.

5.6 Summary

This chapter showed that LSTMs can successfully predict future events from past events, even over very long ranges. This is a necessary precondition for the approach being built in this thesis to function correctly, so getting confirmation that it works is good.

However, the linking algorithm proposed in Chapter 4 no longer seems appropriate, since it is in fact *too* willing to add events to activities. This necessitated the introduction of a new, binary-based linking algorithm. As noted in the previous section, visualising the output shows clusters of new events forming around activities. This is encouraging, and seems to suggest that the method finds activities. However, it would

seem that these activities aren't being seen or enlarged by subsequent layers of the hierarchy. This is a problem that will have to be overcome if this approach is to be a viable method of activity discovery. The next chapter will outline some of the steps taken to address this problem, and subsequently re-introduce binary links.

Chapter 6

Detecting Transitive Activities

The previous two chapters documented the development and evaluation of two activity discovery systems which work by building links between activities based on predictions produced by a neural language model (NLM). The specific model presented in Chapter 4 used a feedforward language model based on word embeddings, as proposed by (Bengio et al., 2003). In Chapter 5, this was replaced with an LSTM-based model, and significant changes were introduced to the algorithm used to construct links. These changes made the algorithm significantly more complex, and only allowed for the construction of binary links (between two events). In this chapter, transitive link-building is re-introduce to the LSTM-based system. Since the previous chapter verified that the language model is predicting future events with a high degree of accuracy, it appears likely that the remaining issues relate to the link building algorithm. Thus, a number of changes to how it works are presented in this chapter. The result is a system that can construct transitive activities

in a manner that is uniquely suited to interleaved datasets. This work has been published as Rogers et al. (2020b).

Section 6.1 outlines the new transitive linking approach. The experiment that was carried out to test the system is described in Section 6.2 and the results of this experiment are presented in Section 6.3. The chapter closes with a summary in Section 6.4.

6.1 Re-introducing transitive links

The revised system introduced in this chapter utilises the same LSTM-based language model as outlined in the last chapter. However, a number of major changes are made to the link building algorithm. These changes are thus emphasised in this chapter, although it will start off by providing a high-level recap of the entire system's operation, in order to help orient the reader.

The notation remains consistant with that used in the earlier chapters of the thesis. From the perspective of somebody using the system, it is assumed that the input is a dataset consisting of a finite series of discrete sensor events $D = \langle d_1, d_2, \ldots, d_L \rangle$. Each individual sensor event has an associated *type* from a fixed set of types T. The type of d_i is written as $t(d_i) \in T$. The primary output from the system is a series of discovered *activities* $\langle Act_1, Act_2, \ldots, Act_N \rangle$, where each activity is a tuple of the form $(Indexes_{Act_j}, Type_{Act_j})$. $Indexes_{Act_j}$ is a set of indexes into the

dataset D, $\{x_1, x_2, \ldots, x_{ActSize(j)}\}$, of length $ActSize(j)$, which can be different for each activity output by the system. $Type_{Act_j}$ is a *type* associated with the discovered activity, analogous to the type of a sensor event discussed above. Act_j and Act_k may share the same type, but they may not share the same set of indexes.

The basic internal operation of the model proceeds according to the following three steps:

- A probabilistic model is build by analysing the dataset. Given a subset of the dataset $D_{i:i+n}$, which is called the *sliding window*, the model can be used to predict the probability distribution over sensor events $P(d_{i+n+l}|D_{i:i+n})$ for all $l \in \{1, 2, \ldots, m\}$. The subset of the dataset $D_{i+n:i+n+m}$ is referred to as the *lookahead window*, m the *lookahead length*, and l the *lookahead offset*.

- The probabilistic model is then used to construct *links* between sensor events if the model is confident that one event can be predicted from the other. Links are grouped together to create the $Indexes_{Act_j}$ part of the output described above.

- Activities are then *clustered* together based on the *similarity of the sensor types present within them*. The clusters are used as the $Type_{Act_j}$ part of the output described above.

The probabilistic model used is a set of m LSTMs, one for each offset in the lookahead window, as described in Section 5.1. The other two

steps, however, work quite differently to those presented in Chapter 5, and are closer to those from Chapter 4. The remainder of this chapter will focus on these two steps.

6.1.1 Building links into activities

Building links between activities is carried out in two stages: building simple binary links, and grouping the links together to form activities. The binary links are built using NLMs. As in the previous two chapters, the networks are first run over the entire dataset. A binary link is then built between the final event in the sliding window and the l^{th} event in the lookahead window if the l^{th} LSTM successfully predicts the l^{th} event from the sliding window (for example, between events B and D in Figure 6.1, which is reproduced for the reader's convenience from the previous chapter, if LSTM 2 predicted event D as the most probable event at position 2 in the lookahead), where $l \in \langle 0, 1, \ldots m-1 \rangle$ indexes over the LSTMs. Doing this naively would make the system vulnerable to building links between common events. In the NLP community, this approach is usually solved by removing common words (stop words), but this could reduce the quality of the resulting activities discovered.

Therefore, as in previous chapters, the system does not work with probabilities directly, but rather with *probability deltas*. Each probability distribution produced by each LSTM model is converted into a distribution of probability deltas instead, much like was done in Chapters 4 and

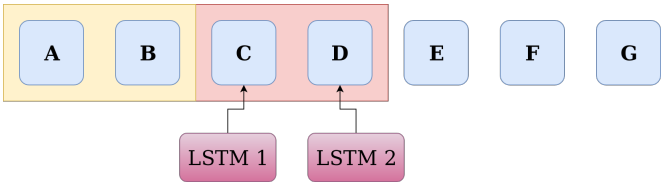


Figure 6.1
In this example, events A and B are inside the sliding window, and are fed as input into two LSTMs. Each LSTM has to predict a probability distribution over the corresponding offset within the lookahead window containing C and D.

5.

6.1.2 Clustering activities into types

Once all of the activities in the dataset have been discovered, the next step is to cluster them by type. This type will then constitute the $Type_{Act_j}$ part of the output described above. A number of clustering algorithms were tried for this, but this comparison found that the type used has surprisingly little effect on the performance of the system. Thus this chapter uses a simple, quick clustering method: two activities can be said to have the same type as each other if they share at least 50% of the same event types.

6.1.3 Building hierarchies of activites

As in the previous chapters, the system presented in this chapter continues to build hierarchies of activities. This process is unchanged from previous chapters: the events from discovered activities are removed from the dataset, and replaced with new abstract events representing the activities as atomic events in their own right. This new dataset is

then used as input for the system, which is run again. This allows rich hierarchies of aggregate activities to be discovered in the dataset.

Figure 6.2 presents a possible output from such a process. If events B, D, F and G are all found to belong to the same activity, they can be removed and replaced with a new event representing the discovered activity. The system can then be trained and run again, which allows this activity to be detected as belonging to other activities. This allows for the building of complex hierarchies of activities, such as the one described in the previous paragraph.

Note that the discovered activity includes events that are not adjacent to each other in the original dataset. For example, event B does not directly neighbour the other events in the activity. This is one of the major strengths of this approach: it does not assume or require that the activities discovered are contiguous. This provides a means to deal with one of the must pressing issues in the field of activity discovery, which is that of *interleaving*, where the person or people under observation are carrying out multiple activities in parallel. From the viewpoint of an activity discovery system, interleaved activities usually look like a person switching back and forth between activities, in much the same way that a modern operating system context switches between running processes to allow multitasking. This approach aims to explicitly address interleaving by disentangling interleaved activities from each other.

One non-obvious aspect of the process shown in Figure 6.2 process is

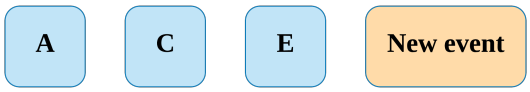


Figure 6.2
If events B, D, F and G are all part of the same activity, these events can be removed from the dataset and replaced them with a new activity. One can then train and run the system from scratch again, allowing the building of rich hierarchies of activities.

why the new event was placed after activity E. For example, event B was also part of the discovered activity the event is replacing, so would it not make equal sense to place the new event between A and C? The placing of new events is decided based on the number of events it removes from various locations of the original dataset. The event in Figure 6.2 removed one event between A and C (event B), one between C and E (event D), and two after E (F and G). Thus, the new event is placed after event E.

6.2 Experiment

As in Chapter 5, the system presented in this chapter was tested using both the SCARE and Kyoto 3 datasets. The system in was again implemented Python using the TensorFlow (Abadi et al., 2015) machine learning library. The neural language model used consists of four stacked LSTM layers, with 150 cells per layer for the lowest level of the hierarchy. As the system ascended the hierarchy it was found that the size increase of the vocabulary due to the addition of discovered activities was straining the network. As a result, increased the size of the network was increased by 50% for each level: level 1 had 150 LSTM cells, level 2

Level number	Average precision
Level 1	0.7694
Level 2	0.8183
Level 3	0.8283
Level 4	0.8342

Table 6.1
Average precision score achieved for a 4-level hierarchy (Kyoto dataset)

Level number	Average F1
Level 1	0.4231
Level 2	0.5427
Level 3	0.6512
Level 4	0.7815

Table 6.2
Average F1 score achieved for a 4-level hierarchy (Kyoto dataset)

had 225, level 3 had 337 and so on. A number of window and lookahead lengths were tested, and it was found that the system performs best when the length of both windows is around 10 to 15 events, as discussed in the next section.

6.3 Results and performance

Compared both to the previous language modelling systems presented in Chapters 4 and 5, and even to the state-of-the-art in the field of activity discovery, the system presented in this chapter shows extremely promising results. Using the Kyoto 3 dataset, given a window length n and lookahead length m of 10, and building a hierarchy of 4 levels, the average precision per level is shown in Table 6.1 and the average F1 score per level is shown in Table 6.2.

Level number	Average precision
Level 1	0.8417
Level 2	0.8795
Level 3	0.9001
Level 4	0.9111

Table 6.3
Average precision score achieved for a 4-level hierarchy (for the SCARE dataset)

Level number	Average F1
Level 1	0.5129
Level 2	0.6588
Level 3	0.7425
Level 4	0.7902

Table 6.4
Average F1 score achieved for a 4-level hierarchy (for the SCARE dataset)

Similarly high performance is obtained when running on the SCARE dataset, as shown in Tables 6.3 (for the precision) and 6.4 (for the F1 score).

If one compares the results of this system with the LSTM system from Chapter 5, it can be seen that the average precision of this new system on the SCARE dataset is (after four layers of training) 91%, compared to an average of 66% with the system from Chapter 5. Similarly for the Kyoto dataset, the average precision of the new system is (again, after building a four layer hierarchy) 91%, compare to about 70% for the system from Chapter 5. These results are in line with what most real-world activity discovery systems can achieve, despite the fact that such systems generally use far richer input data: for example explicit temporal information, which this system does not require.

It can be seen that the results improve the higher up through the

Event type	Precision
new_event_45	1.0
new_event_46	1.0
new_event_47	0.75
new_event_48	1.0
new_event_49	0.5
new_event_51	0.5

Table 6.5
Extract of the full results, showing the precision of each activity type found

hierarchy the system ascends. This is expected, since the events become more abstract and thus closer to the (very abstract) activities in the ground truth. The results were also computed for each discovered activity (the results presented in Table 6.1 is the average over these scores.) These are too large to be presented in full in this thesis, but an extract of the full results are presented as Table 6.5.

A particularly interesting and encouraging result is the difference in performance when dealing with interleaved and non-interleaved datasets. Table 6.6 shows the average precision for the system when given a *non-interleaved* dataset as input. The Kyoto 3 dataset actually consists of two datasets, one containing interleaved and another containing non-interleaved activities. The same types of activities are present in both datasets, but the volunteers carrying them out were asked not to interleave them as they carried them out for the non-interleaved dataset. Compared to the results for the interleaved dataset (Tables 6.1 and 6.3) it appears that the lack of interleaving is actually confusing the system, which is the opposite to what is usually observed in activity discovery

Level number	Average precision
Level 1	0.7376020200520275
Level 2	0.7744498540343416
Level 3	0.7964471494200084
Level 4	0.8018082799378542

Table 6.6
Average precision score achieved when using the non-interleaved Kyoto dataset

systems. This gives a strong reason to claim that this system is well suited to dealing with interleaved datasets. It also means that performance could likely be boosted further by an ensemble of this system and traditional activity discovery systems, since these results demonstrate that they arguably have different strengths, and combining models with different strengths is generally a good idea when carrying out ensemble learning.

A number of different hyperparameter values were also tested on the system. In particular, the affect of adjusting both the window and lookahead lengths was studied. Increasing the window length has a moderate negative impact on the observed results, as shown in Table 6.7. This indicates that the extra information provided in the longer sliding window actually ends up confusing the LSTM networks, since they observe conflicting signals as a result of now having multiple activities visible in their input at any one point in time. Most activities, even when highly interleaved, tend to be very “local", with events that constitute the activity being located quite close together in the dataset.

A strong negative correlation was also observed between lookahead

Window length	Average precision
10	0.80
15	0.80
20	0.79
25	0.77

Table 6.7
Relationship between window length and precision (for Kyoto)

Lookahead length	Average precision
10	0.80
15	0.64
20	0.56
25	0.48

Table 6.8
Relationship between lookahead length and precision (for Kyoto)

length and precision, shown in Table 6.8. This is actually somewhat surprising, since one would expect that a system which deals so well with interleaving would be confident about linking distant events together. Having said that, the further apart two events are, the less likely they are to be connected, even when dealing with an interleaved dataset, so this arguably isn't that unusual. Nonetheless, the extent of the negative correlation is worth pointing out.

Minimum description length (MDL) has already been mentioned a number of times in this thesis (see Section 2.5 in particular). Cook et al. (2013) suggest using this as the basis for another metric for activity discovery systems, namely that of *compression ratio*. Since the system presented in this chapter is constructing a hierarchy of activities by removing the sensor events that are found to belong to the discovered activities, the dataset reduces in size over time. Compression ratio alone

can serve as a metric, since having a high compression ratio can be a sign that the system is correctly finding activities present in the dataset. This system compresses the original Kyoto3 dataset to around 36% if its original size, which compares extremely favourably to the 68% achieved by the system in Chapter 5 (lower compression ratios indicate higher compression).

Cook and Krishnan (2015) proposes that the concept of compression ratio could be converted into a more principled metric by measuring how well compression ratio *generalises*. In traditional machine learning, one is generally more concerned with how a system deals with novel input compared to how it deals with input seen in the dataset. This allows one to be sure it is learning a signal present in the dataset, rather than just memorising the contents of the dataset. This is typically measured using techniques like holding out a validation and test dataset from the main dataset. If a system is generalising well, its performance on the testing dataset should be similar to the performance on the validation dataset. Likewise, if an activity discovery system compresses the dataset by a certain amount, it should also compress a testing dataset by the same amount. The system was tested using ten-fold cross-validation. The results, presented in Table 6.9, show clearly that the system is finding activities that generalise well to the test dataset. Note how favourably this compares to the compression ratios presented in previous chapters: the results consistently show a reduction to 30-40% of the original size of

Cross-validation	Compression ratio
Fold 1	0.3927
Fold 2	0.3290
Fold 3	0.3363
Fold 4	0.3469
Fold 5	0.3532
Fold 6	0.3862
Fold 7	0.3154
Fold 8	0.3260
Fold 9	0.4082
Fold 10	0.3371

Table 6.9
Ten-fold cross validation of the compression ratio produced by the system (for the Kyoto dataset)

the dataset, compared to about 80% or so for the two previous systems.

6.4 Summary

This presented an activity discovery system based on a language model that is capable of discovering complex hierarchies of interleaved activities in activity discovery datasets. The results show that this system is capable of matching or even outperforming the state-of-the-art, a claim evidenced by the results presented above.

Achieving about 80% or so on a performance metric is quite a common achievement in the activity discovery community, which justifies claiming that a working activity discovery system based on the link-building recurrent language model that was introduced has been presented in this chapter. It has been shown that this is a viable approach to activity discovery, even for complex datasets which are actually more complex than those typically used in this problem domain, in particular

by being heavily interleaved and lacking true temporal information.

One thing that does remain to be done, however, is to provide some discussion on the *evaluation* of activity discovery systems. The systems presented were tested by evaluating their performance on a number of metrics, since it seems that no one metric can provide a complete picture of how well the system is performing. The question of how to fairly evaluate activity discovery systems still appears to be an open one, and thus that will be the topic of the next chapter.

Chapter 7

Evaluating Activity Discovery Systems

Evaluating activity discovery systems is a challenging task. This is something that has already been alluded to in previous chapters. This chapter will discuss these issues in further detail. In particular, it will focus on the limitations of existing evaluation metrics and metrics that have already been proposed in the AD literature. The particular approach to activity discovery proposed in this thesis also introduces additional complications to evaluation, and these will be addressed along with proposals about how to mitigate them. Previous chapters have alluded to the idea that *differing levels of abstraction* represent a major challenge in evaluating AD systems. This is a particular focus of this chapter.

This chapter differs from previous contributions in this thesis in that it is not focused on a single concrete proposal. At present, no single evaluation metric gives a completely unbiased and accurate reflection of an AD system under evaluation. Thus a major underlying assumption of this chapter is that AD systems should be evaluated using a range of

metrics, and the performance of one single metric may not be sufficient to give an accurate view of a system's performance.

This chapter begins by outlining some of the existing approaches for evaluating activity discovery systems that have been presented in the literature (Section 7.1). In Section 7.2, it is hypothesised that differing levels of abstraction are a major reason for some of the weaknesses of existing evaluation metrics, and a tentative solution to this problem is presented and used to assess the performance of the topic modeling based model from Chapter 3 and the language modeling based approach from Chapter 6.

7.1 Evaluation metrics in the literature

Cook and Krishnan (2015) provide a good overview of existing approaches to evaluating activity discovery systems, and much of the following discussion takes this as a starting point. Specifically, Cook and Krishnan (2015) propose a taxonomy of evaluation metrics which classifies metrics into one of four fundamental types. These are *human evaluation*, *predictive stability*, *compressive stability* and *clustering evaluation techniques*.

The given taxonomy is supplemented by adding the concept of *minimum description length*, an important concept derived from information and algorithmic information theory which is highly relevant to this dis-

cussion. Minimum description length has already been introduced in Section 2.5.

7.1.1 Human evaluation

One approach to evaluating activity discovery systems is to simply rely on human volunteers to carry out the task of evaluation (Hammid et al., 2012). Although manually inspecting the output from the system can be a useful debugging tool, it is not a reliable way to evaluate in the general case. In Section 7.2, it will be argued that differences in levels of abstraction between the outputs of AD systems and ground truths considerably increase the difficulty associated with evaluating AD systems fairly. A human evaluator could therefore score an AD system poorly because the abstractions that they see in the dataset do not match the abstractions output by the system, even if those abstractions are actually present in the dataset. Human evaluations are therefore likely to be non-reproducible between individuals, and can fall victim to assumptions about the “correct" abstractions held by the human annotators. For these reasons, human evaluation will be mostly ignored in this chapter. The evaluation metrics used in this thesis assume that unbiased, automated and repeatable evaluation metrics are always preferable to subjective evaluations.

7.1.2 Predictive stability

When evaluating any machine learning system, one is usually interested in determining the degree to which the learned model is *generalised* (that is, the degree to which it can be applied to similar but unseen data). *Predictive stability* is proposed as the analogous concept for activity discovery. A machine learning system can be verified to have generalised by holding out a small subset of the dataset as a *test set*, and use the remainder of the dataset as a *training set*. If a model is sufficiently general, it is expected that the performance on both datasets will be similar. Analogously, an activity discovery model can be expected to find the same types of activities in roughly the same quantities in both the training and test datasets. Such a model is said to exhibit high predictive stability. In more formal terms: before the activity discovery model is applied to a dataset D, D is first split into two subgroups B and C, such that $D = B \cup C$, and usually also so that $|B| \ll |C|$. C is then used as a training set, so the discovered activities become $Y = g(C)$. One can then see how well the activities Y generalise to B by seeing can they be observed in B with the same average frequency as in C. If so, it can be said that the model predicts activities that are stable across the dataset. It would seem, however, that this metric has not found wide use. There may be a number of reasons for this, including lack of awareness brought about by the small size of the activity discovery research community.

7.1.3 Compressive stability

The argument just made leads naturally to the next evaluation method, the related idea of *compressive stability*. An AD system can be thought of as a compression algorithm, in the sense that it is providing a way to reduce the size of the raw dataset. Consider a scenario where there is a dataset D consisting of 6 sensor events labeled from $Event_1$ to $Event_6$. Suppose that after being passed into an AD model, events $Event_2$ and $Event_3$ are found to constitute a single activity. These two events are therefore removed and replaced them with a single "New event". It can thus be said that the dataset was compressed from a length of 6 sensor events to a length of 5 events. This fact could be utilised directly, and performance could be measured in terms of *compression ratios*. Compressive stability is a more formal approach to this basic idea. The dataset is again split into subsets $B \subset D$ and $C \subset D$, the activity discovery system is trained on set C and the compression ratio achieved on set C is then measured. Then, the process is repeated on set B – if the activity discovery system has successfully generalised the training data it is expected that the resulting compression ratio for set B will be similar to the compression ratio of set C.

Perplexity, a measure proposed by Jelinek et al. (1977) and already in wide use in the machine learning community, is an example of a measure of compressive stability. Perplexity is often used, for example,

to measure the performance of *language models*, which compute the probability of natural language sentences. Given a natural language sentence $X = \langle x_1, x_2, \ldots, x_N \rangle$ of words x_i and a language model M, the perplexity of X is defined by Equation 7.1. Here, H is information theoretic entropy, and $P_M(X)$ is the probability of X according to the model M.

$$Perplexity(X) = 2^{(H(P_M(X)))} \tag{7.1}$$

7.1.4 Minimum description length

Compressive stability relates very closely to the concept of *minimum description length* (MDL) (Rissanen, 1978), which has already been introduced in Section 2.5 and mentioned a number of times throughout this thesis. MDL can be viewed as trying to solve a similar problem to compressive stability, but rather than just measuring the compression ratio (or even the stability of the compression ratio), it works by measuring the length of the optimal bitstring required to represent the dataset after compression by the model. For the reader's convenience, the MDL equation (initially presented as Equation 2.8 on page 46) is duplicated below as Equation 7.1.4 below. Recall that $L(M)$ denotes the size of the model M, and $L(D|M)$ is the length of the dataset D under the compression of model M.

$$MDL(M, D) = L(M) + L(D|M) \tag{7.2}$$

Lower values indicate better performance in this case. This can be made clearer with a concrete example. Recall that every member of the dataset D is an event drawn from an alphabet Σ and L is the length of D. If $|\Sigma| = N$, one can see that the length of the dataset in bits should be around $L \times \lceil log_2(N) \rceil$. This number could be reduced by using some optimal encoding scheme (i.e. so that frequent events get a shorter binary representation). In that case, the dataset will be of length:

$$L(D) = \sum_{d \in D} -log_2(P_d) \tag{7.3}$$

where P_d is the probability of symbol $d \in \Sigma$ in the dataset (i.e. the number of occurrences of d in D divided by L). This is the minimum description length of the dataset without any model at all.

Introducing a model should allow the length of the bitstring $L(D)$ to be reduced to $L(D|M)$, since many of the events will have been replaced by new events representing activities. Of course, there is no point in doing this if the model needed to compute this compressed dataset is very large. Thus there has to be some way to compute the length of the model's bitstring representation, $L(M)$. The way this is computed will vary depending on the nature of g. For example, if g is a neural network, it could be represented as the length of the binary values of the weight matrices required to specify the network. For each extra bit of information added to the model, one can expect to save more than 1 bit (on average) from the dataset representation. A model which cannot

do this has not compressed the raw dataset, and thus is regarded as not useful, at least from an MDL perspective (Barron et al., 1998).

7.1.5 Clustering evaluation techniques

Returning to the metrics suggested by Cook and Krishnan (2015), a large body of work on *clustering evaluation techniques* exists also, and can be employed for evaluating activity discovery. These are generally based on the intuition that the distance between elements of a cluster (the *intracluster distance*) should be minimised, while the distance between clusters (the *intercluster distance*) should be maximised. Cook and Krishnan (2015) propose using clustering techniques to validate the activity types produced by an activity discovery system. Many clustering evaluations could be employed: one example is the *Dunn index* (Dunn, 1974). For each pair of clusters i and j, the Dunn index is the ratio of the smallest intercluster distance over the largest intracluster distance. For m clusters, the Dunn index is defined as:

$$DI_m = \frac{\min\limits_{1 \leq i < j \leq m} d(i,j)}{\max\limits_{1 \leq k \leq m} f(k)} \tag{7.4}$$

where d is a valid intercluster distance, and f a valid intracluster distance. Note that the index is agnostic to the specific d and f distance metrics used.

Another common cluster evaluation technique is the *Silhouette index*

(Rousseeuw, 1987). This works on two directly defined functions:

- $a(i)$ which measures the intracluster distance by averaging the distances between item i and all other members of the cluster that i belongs to, and
- $b(i)$ which measures the intracluster distance by averaging the distances between item i and all other members of all clusters that i does not belong to.

The Silhouette index, $s(i)$, for a single item i, is then defined as:

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))} \tag{7.5}$$

These two metrics can be used to determine if the activities discovered by an activity discovery system are coherent, and if it makes sense to give them the same type. However, they cannot be used to evaluate the overall performance of the system.

7.2 Abstraction issues in ground truth metrics

While reading Section 7.1, one notable fact is that that all of the metrics proposed seem to refrain from using ground truths as a gold standard with which to compare the output of the AD system under evaluation. One would presume that a simple and effective way to evaluate would be to simply compare the output of the AD system to a known ground truth

(the “right" answer, so to speak) and measure how closely they relate, perhaps by use of an F-measure if raw accuracy is unsatisfactory. Indeed, this is the systems presented in Chapters 3 to 6 were evaluated. Although this is a perfectly valid way of evaluating machine learning models in the general case, there are two major grounds to be suspicious of ground truth comparisons for AD. The first of these is that, by definition, an AD system must be *unsupervised* – that is it trains without making use of any sort of output labels in the dataset. The entire point of the *discovery* of activities is to provide a way for the detection of plausible activities in *unannotated datasets* without any ground truth. In a real-world use case, it is quite possible that the model will therefore be trained on a dataset for which no ground truth to compare against exists, and so a means of evaluation that can be used even in these kinds of situations must be found.

The second issue with ground truth-based evaluation is the *subjectivity inherent in the output of any AD process*. Although the behaviours reflected in the sensor stream may be objective and leave no room for subjective interpretation, the same cannot be said for the activities detected in the stream. For example, the point at which an activity can be said to start and end is arbitrary. Consider the hypothetical case of a sensor stream in a house where an activity corresponding to *making dinner* takes place every evening. One could say that this activity begins when the resident(s) of the house enter the kitchen to cook, or when

they turn on the oven, or when they first put food into the oven. Different AD systems (and indeed human annotators) may well use different boundaries for their activities in this manner, and one cannot privilege one annotation over the other. By extension, it is also possible for entire activities to be (in a sense) subjective. For instance, what if one argues that the resident entering the kitchen to cook does not constitute part of the *making dinner* activity, but rather an activity in its own right, perhaps called *preparing to make dinner*? This issue provides major challenges for the evaluation of these systems. Note that it is a very particular kind of subjectivity been spoken about here. Intuitively, any system that fails to find a consistent activity every evening around dinner time in the hypothetical house seems to be objectively wrong in some way, since it cannot pick up a real pattern that exists in the data. But aspects of the pattern (its size, constitution, cardinality and so forth) are subjective in a way that makes comparison to a ground truth seem like an inherently unfair approach to evaluation.

At the time of writing, very little has been published on this problem in the specific case of AD, although some work has been done on related problems (Ward et al., 2006). For example, a number of publications exist that address the issue of evaluating systems that have to predict when a temporal action begins and ends. Some examples of this include the activity *recognition* (not discovery) problem – such a system might successfully detect that an activity has begun, but the start and end times

may not align perfectly with the ground truth.

One such publication that is of particular interest is Ward et al. (2006). This paper does not propose an evaluation metric, but rather an *error analysis method*, in other words a means to detect the types of errors that a system under analysis seems to make consistently. The analysis proposed in the paper is conceptually quite simple, and works by enumerating the occurrences of a number of different error types that the paper's author identified. A diagram illustrating the approach is shown in Figure 7.1. The input to the error analysis process is one or more *channels* of output-ground truth pairs. The metric splits this input into a number of *segments* (denoted by dashed lines in the diagram), which are contiguous time slices in which both the output and ground truth for all channels remains static. This means that at most one error can occur in each segment, so errors can be checked for on a per-segment basis. The actual error analysis consists of three parts, the first of which is called *event scoring*. Here, four types of errors are identified and labeled. An *insertion* error (denoted by the letter I on the diagram) indicates that an event exists in the output, but does not have a corresponding ground truth, and so has been erroneously inserted by the system. The opposite situation, where an event is seen in the ground truth but is missing the system output, is called a *deletion* error, and is denoted by the letter D on the diagram. Another type of error counted in the event scoring stage is called *merge* (M on the diagram), and is characterised by an output

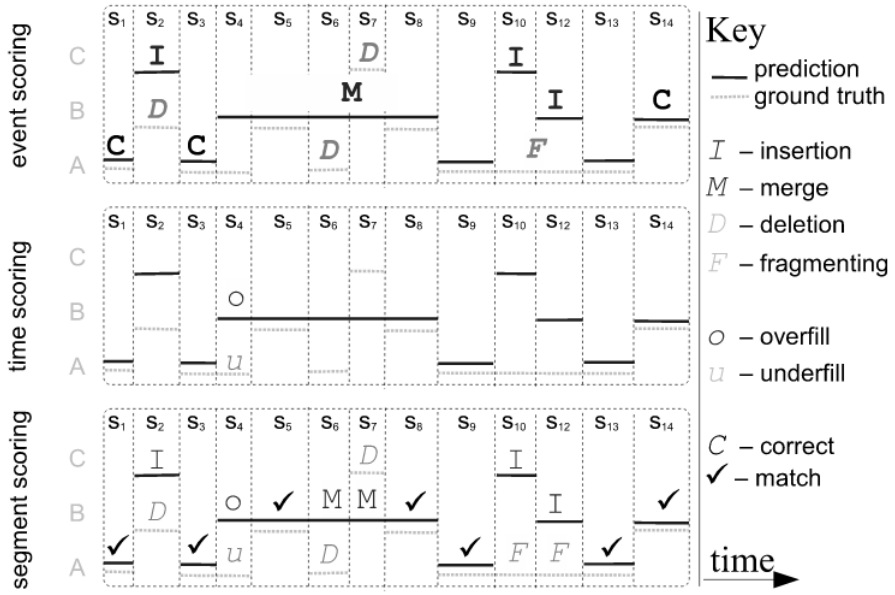


Figure 7.1
An example of the error analysis proposed by Ward et al. (2006), showing insertion (I), deletion (D), merge (M), fragmentation (F), underfill (u) and overfill (o) errors. Correct is denoted by the letter C, and matches are shown on the diagram as tick marks. Taken from Ward et al. (2006)

that detects multiple distinct ground truth events as a single output event (i.e. where several ground truth events are merged into a single output event). Again, the next error type, called *fragment* (F in the diagram) is the opposite of merge. Here, a single ground truth event appears as two or more distinct events in the output. Some segments will not show any of the four error types. These can be marked as *correct* (C on the diagram).

The second stage of the evaluation metric is called *time scoring*. Again, it is carried out on a per-segment basis. Here, there are two types of errors detected that would have been missed by the event scoring stage. These errors are referred to as *overfill* (o) and *underfill* (u)

respectively. An overfill error occurs when an output event matches a ground truth event, but it extends beyond the boundaries of the event as viewed from the ground truth. An underfill error occurs when the output and ground truth events match, but the boundaries of the ground truth extend beyond the boundaries of the output event.

The final stage of the error analysis method, called *segment scoring*, consists simply of combining the results of the last two stages. Thus, all segments are given a single label. If one of the six error types (insertion, deletion, merge, fragment, overfill or underfill) are observed to occur in that segment, then the entire segment is labeled with that error type. If no error is found, then the entire segment is labeled as a *match* (shown as a tick mark on the diagram). The matches denote the regions of the output where the model under evaluation is indisputably correct. The other segments contain a classification of the type of error present. For reasons that will be discussed later in this chapter, it may not be fair to call some of these *errors* in the truest sense of the word, since they could be differences of opinion. The authors of the paper propose summarising the results of the analysis into a table to allow people to see the error types, without merely presenting a single scalar value. Interested readers are referred to the referenced paper by Ward et al. (2006) directly for a more detailed introduction to the error analysis method.

7.2.1 Formalising abstraction

Many of the subjective differences outlined above that affect ground truth-based metrics can be *attributed to differences in the level of abstraction that the various systems under evaluation are outputting*. Before going any further, it is worth clearly explaining what is meant by this.

Supposing that g and h are activity discovery models, Y_g is the set of activities output by g (where each $y \in Y_g$ is a subset of D), and likewise Y_h is the set of activities output by h (where each $z \in Y_h$ is a subset of D). This scenario can be formally represented as:

$$g(D) = Y_g \tag{7.6}$$

$$h(D) = Y_h \tag{7.7}$$

Suppose that for a particular $y \in Y_g$ and a particular $z \in Y_h$, it is found that $y \subset z$, in other words y is strictly a subset of z (i.e. $\forall i(i \in y \Rightarrow i \in z)$, but $\exists i(i \in z \wedge i \notin y)$). This means that z is thus a *more abstract* version of the activity y: everything in y is also in z, but the reverse is not true. To make this more concrete, one can imagine y being an activity like *making dinner*, and z being a more abstract version of the same activity like *having dinner*, which contains *making dinner* in its entirety, in addition to other sensor events covering the consumption of the dinner, and perhaps cleaning up after. Notationally, this scenario

is represented as $y \prec z$, which can be read as "y is less abstract than z", or "y precedes z".

To complicate matters further, it is important to resist any temptations one may have at this point to claim that AD model g is less abstract than h, simply because it output a less abstract activity in one instance. It is entirely possible that multiple levels of abstraction are interleaved in the output of the models, i.e. it may be possible to find activities for which g finds a more abstract version than h. Unless all activities found by g are less abstract than or equal to all activities found by h, one should refrain from talking about the abstraction of entire models.

7.2.2 Abstraction-aware evaluation

The abstraction issues discussed above make developing a single, reliable metric for an activity discovery system very difficult. This is because activities and any ground truth used could well be at different levels of abstraction, meaning they cannot be fairly compared. As noted at the beginning of this chapter, this suggests that an ensemble of metrics be used together, rather than concentrating on one in particular. Evaluation metrics can also be supplemented by modifying the output produced by activity discovery systems to convert them to a form that is closer to what existing evaluation metrics expect. In particular, the outputs of many activity discovery systems tend to be *fragmented*, in which one long activity that contiguously spans a large number of events is

detected by the activity discovery system as multiple small activities of the same type. Rogers et al. (2018) proposed a simple way of dealing with fragmented output, which expands the range of possible metrics available for such a sample.

Recall that D is a dataset, and G is the associated ground truth. Assume $|G| = |D|$, and that each element $g \in G$ is a vector of k Boolean values, where k is the number of activities in the ground truth. Thus, $G_{i,j}$ is true iff activity j is true or active for the i^{th} event in the dataset. This discussion will also commit to a specific structure for the output Y, since not doing so would make the formalism needlessly abstract. The formalism can be easily adapted for other output structures and formats, although no attempt is made to prove this here. Y is modelled as a matrix, such that each value $Y_{i,j}$ represents the probability that activity j is true or active for the i^{th} event. Given a particular index into the ground activities g, an index into the output activities y, and a real-valued threshold value t, the true positives of the AD system are then defined to be:

$$TP_{gyt}(G, Y) = \sum_{i=1}^{L} \mathbb{1}(G_{i,g} \wedge Y_{i,y} \geq t_y) \tag{7.8}$$

where $\mathbb{1}$ is an indicator function that evaluates to 1 if the Boolean formula passed to it is true, and 0 if it is false, and $G_{i,g}$ will equal 1 if the g^{th} activity is active for the i^{th} event in the dataset. Thus $\mathbb{1}$ will evaluate to 1 for the i^{th} event in the dataset if the ground truth activity g was

active according to the ground truth, *and* the output activity y was active according to the AD system. An output activity can be defined as being active if the probability that it is active at event i exceeds a threshold t_y. The vector of thresholds t is a meta-parameter, and one can compute a different value of t_y for each proposed activity in Y. With minor modifications, this process could be used to compute false positives and true and false negatives also. From here, one can straightforwardly calculate the F-measures for the system.

A diagrammatic example of this proposal is shown in Figure 7.2. Here, each of the horizontal lines labelled A to E represent a single channel of information. Channel A is a ground truth, as found in an annotated dataset. Channel B represents the *raw output* of an AD system for a particular event type. The output overlaps to a degree with the ground truth. The degree of overlap can be improved by extending the length of channel B to match channel A, as shown in channel C (which is the *extended output*). This can be formalised by modifying Equation 7.8 as follows:

$$\begin{aligned} TP_{gyt}(G, Y) = \sum_{i=1}^{L} \mathbb{1}((G_{i,g} \vee g \in G_{i-1,g} \vee G_{i+1,g}) \\ \wedge (Y_{i,y} \geq t \vee Y_{i+1,y} \geq t_y \vee Y_{i-1,y} \geq t_y)) \end{aligned} \tag{7.9}$$

Rather than strictly requiring that the probability of activity y during event i at least meets the threshold, one can look to the events immediately before and after the current (i^{th}) event, and also accept event i as a

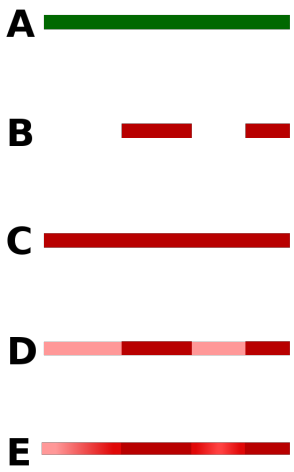


Figure 7.2
If channel A is an output, and channel B is the output from a system under evaluation, B is extended to match A (*extended output*, channel C), optionally making the extensions values less than 1. In channel D (*staircase output*) a smaller value is used for the extensions, and a reducing gradient is used for channel E (*gradient output*).

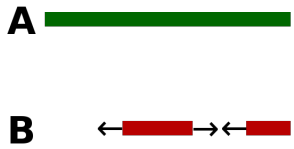


Figure 7.3
In summary, he unfairly negative effect that fragmentation has on activity recognition systems is solved by incrementally expanding discovered activities (channel B) to the left and/or the right (as denoted by the arrows) for as long as doing so causes the F-1 score computed against the ground truth (channel A) to increase.

valid true positive if one of its neighbours are also a true positive. The ground truth is similarly extended in the same manner. This computation is then repeated *as many times as needed for the true positive value to stop increasing*. Thus, the length of both the ground truth (channel A in the diagram) and the output are extended until their respective lengths match. This is essentially equivalent to extending the size of the activities to fix the fragmentation issue automatically, as shown in Figure 7.3.

One potential criticism of the presented proposal is that it is making the evaluation *too easy* for the activity recognition system. For this reason, channels D and E can be used as alternatives to C for comparison to the ground truth (channel A). In these cases, the darkness of the colour corresponds to its magnitude, with the number 1 being as dark as channels B and C, and lower numbers (closer to zero) being represented with a lighter number. Most evaluation metrics (raw similarity, F-measures and so on) work by counting the number of matches between two binary channels. For example, F-measures build a confusion matrix, and match a False from both the ground truth and output channels as a true negative, a False from the ground truth and True from the output as a false positive and so on. This evaluation technique can thus use non-binary, fuzzy values instead of these binary comparisons, so that the counts for the confusion matrix are incremented by a number between zero and one. For channel D (the *staircase output*), a value of 1 is used for the true overlaps, and a smaller value (0.4, or $\frac{2}{5}$ in the experiments, see below) for the extensions. Formally, this becomes:

$$\begin{aligned} TP_{gyt}(G,Y) = \sum_{i=1}^{L} \mathbb{1}(g \in G_i \wedge Y_{iy} \geq t) + \\ \frac{2}{5}(G_{i-1} \vee g \in G_{i+1} \vee Y_{(i+1)y} \geq t \vee Y_{(i-1)y} \geq t) \end{aligned} \tag{7.10}$$

There are now 2 indicator functions: the $\mathbb{1}$ function mentioned previously, and a new $\frac{2}{5}$ function, which returns a value of $\frac{2}{5}$ if its input is true, and false otherwise.

Finally, in channel E (the *gradient output*), the extensions do not have a fixed value, but rather have a value of $1 - (0.001 \times n)$, where n is the number of events away from the true overlaps, but cannot have a value below zero. Again, an indicator function is defined which is suitable for this purpose, but rather than calling it $\mathbb{1} - (\mathbb{0}.\mathbb{0}\mathbb{0}\mathbb{1} \times \mathbb{n})$, it is instead given the more succinct (but less descriptive) name $\mathbb{f}$.

$$TP_{gyt}(G, Y) = \sum_{i=1}^{L} \mathbb{f}((g \in G_i \vee g \in G_{i-1} \vee g \in G_{i+1}) \\ \wedge (Y_{iy} \geq t \vee Y_{(i+1)y} \geq t_y \vee Y_{(i-1)y} \geq t_y)) \tag{7.11}$$

Note that Figure 7.2 only shows the extensions applied to the output channels. The ground truth channels should also be extended according to the above process.

At this point, the usual performance metrics used to evaluate ground truth-based systems can be employed. This could include raw percentage accuracy measures, or preferable a more sophisticated metric like F-measures.

7.2.3 Evaluating the metric

In order to try to determine if the proposed metric is useful, an experiment was carried out using the activity discovery system that was presented in (Rogers et al., 2016), and covered in Chapter 3 of this thesis. The interested reader is directed back to this chapter for a detailed explanation of how this system works, but in summary the dataset D is split

up into $L - w + 1$ subsets using a sliding window of length w and each window is then run through a topic modelling algorithm as if it were a single document. This allows probability distribution to be computed over topics for all events in the dataset. These values are thresholded to assign each event to zero or more activities, using the t_y threshold previously mentioned in this section. In effect, this threshold is the prior over activities. For each ground truth activity g and output activity y, a candidate threshold value t_{gy} is computed that comes closest to making $c_{gy} = \|P(g \in G_i) - P(Y_{iy} \geq t_{gy})\|$ (the difference between the ground truth and output activity probabilities) equal to zero. The final threshold t_y is then simply the threshold that has the minimal c_{gy} value over all gs, i.e. $t_y = arg\ min_{t_{gy}}(c_{gy})$. This thresholding gives a dataset of 10 channels, consisting of 5 ground truths and 5 discovered topics (outputs). The F1 score is then computed for each $(ground\ truth, topic)$ pair for each of the 4 types of evaluation shown in Figure 7.2. Each ground truth is then associated *with the single topic that scores highest with it according to the extended F1 score*.

The result of the experiment described above is presented after being run on two different datasets. The first of these datasets was generated by the author using a state machine probabilistically moving from state to state and emitting events, with some events being more common than others for each state. The results of this experiment are shown in Table 7.1 below. The first two columns show the $(ground\ truth, topic)$ pairs,

Table 7.1
Performance metrics gathered by the experiment on an artificial dataset

Topic	Label	F1	Extended F1	Staircase F1	Gradient F1
Activity A	Topic 2	0.6385	0.9521	0.9865	0.9896
Activity B	Topic 3	0.2269	0.9211	0.9834	0.9853
Activity C	Topic 1	0.3159	0.876	0.9619	0.977
Activity D	Topic 4	0.1146	0.8426	0.8923	0.8994
Activity E	Topic 0	0.01835	0.1428	0.8053	0.8192

and the remaining columns show the raw F1 score (i.e. the score calculated *without* using the method from this chapter), the extended F1, the staircase F1 and the gradient F1 respectively. The results show an interesting pattern: for each row, the raw F1 score is substantially lower than the equivalent scores computed with the proposed method. Bearing in mind that the only difference between these metrics are that the latter three take the concept of abstraction into account in the manner described above, this can be taken as evidence that the proposed metrics are a fairer way to evaluate such systems. The raw F1 score is unfairly penalising the system for what could actually be valid disagreements over abstraction levels and the start and end times of activities, while this method does not do so.

In order to evaluate the metric on a more challenging dataset, the experiment was repeated on the *SCARE corpus* already discussed in previous chapters (see Section 3.4). The results of this experiment are presented in Table 7.2. Again, one can see a substantial improvement in performance when the proposed metric is employed. Note that the SCARE corpus is extremely challenging: it is unusual for activity re-

Table 7.2
Performance metrics gathered by the experiment on the SCARE dataset

Topic	Label	F1	Extended F1	Staircase F1	Grad. F1
goal_move_box	Topic 1	0.135	0.644	0.683	0.831
goal_move_rebreather	Topic 5	0.306	0.948	0.958	0.959
goal_move_quad	Topic 2	0.111	0.623	0.563	0.547
goal_move_silencer	Topic 4	0.057	0.581	0.818	0.844
goal_move_picture	Topic 3	0.073	0.513	0.65	0.638
null_goal	Topic 0	0.0	0.0	0.0	0.0

cognition systems to obtain a score greater than about 0.7, let alone AD systems, which must produce their output without access to the ground truth. This metric could not only give a fairer means to evaluate AD systems, but potentially a fairer means to evaluate corpora used also, by highlighting excessively narrowly defined activities in a corpus's ground truth.

Note that the difference in results between the raw F1 score and the score as computed by the proposed method is substantial. When the output from the system under evaluation agrees strongly with the ground truth, the two scores should begin to converge with each other. For this reason, the proposed metric might be better suited as an *evaluation of the degree of fragmentation in the output*, rather than a metric of overall performance.

More concretely, the F-measure computed by the proposed metric minus the standard F1 score can serve as a useful measure of the *degree of output fragmentation*. To see why, recall that in Figure 7.2, the output channel B is highly fragmented. When corrected into the extended output

Table 7.3
F1 metrics gathered by the language model system on the SCARE dataset

Layer	F1	Extended F1	Staircase F1	Grad. F1
Level 1	0.5129	0.6103	0.5248	0.5337
Level 2	0.6588	0.6963	0.6927	0.6746
Level 3	0.7425	0.8579	0.7758	0.8177
Level 4	0.7902	0.8826	0.8497	0.8135

Table 7.4
Precision metrics gathered by the language model system on the SCARE dataset

Layer	Precision	Extended Prec.	Staircase Prec.	Grad. Prec.
Level 1	0.8417	0.8517	0.8435	0.8629
Level 2	0.8795	0.9134	0.9403	0.9841
Level 3	0.9001	0.9152	0.9232	0.9513
Level 4	0.9111	0.9391	0.9165	0.9502

channel C, it can be seen that it now aligns with channel A. Thus, it would achieve an F1 score of 100%. Channel B's F1 score would be considerably smaller, because it has many false negatives. Thus, the F1 score of C *minus* the F1 of B would be quite a large number, indicating a high degree of fragmentation.

The metric was also tested on the transitive language model system presented in Chapter 6. The results of this are shown as Tables 7.3 (for the F1 score) and 7.4 (for the precision).

The results here show a far less dramatic difference between the raw metric scores and the scores as calculated by the metric compared to the topic model system. The reason for this is simply that the language model system presented in Chapter 6 performs better than the topic model system, and as a result there is less fragmentation in the output.

7.3 Summary

This chapter focused on the fair evaluation of activity discovery systems. Activity discovery is a very challenging problem, and thus effective and fair evaluation is very important. Usually a number of metrics should be employed for evaluating any one system – several such metrics were discussed in this chapter, with a particular emphasis on metrics that relate to MDL such as compressive stability. In addition, this chapter has introduced a new evaluation metric, which can serve as a useful contribution to the toolbox of metrics available to activity discovery systems. This new metric was tested by applying it to two of the activity discovery systems presented in this thesis. The results highlight that the metric can handle abstraction issues that most other metrics unfairly ignore. They also reinforce that the activity discovery system presented in Chapter 6 performs well compared to the state of the art. The metric also has a useful purpose for detecting fragmented outputs, and measuring the degree of fragmentation, which is a common side-effect of a number of activity discovery algorithms, and can make them appear to perform worse than they actually do.

Chapter 8

Conclusions

This thesis addressed the problem of activity discovery by proposing and testing a novel hierarchical language model-based activity discovery model. It has been shown that the model can achieve very high performance, and in particular can achieve state-of-the-art performance on interleaved activities, which has historically been a difficult area for activity discovery systems.

8.1 Primary contributions

In Section 1.5, the following list of the primary contributions in the thesis was presented, which has been reproduced below for the reader's convenience:

- A model for the construction of hierarchical trees of activities, where low-level activities are discovered within sequences of events, and higher-level activities are composed of a combination of events

and lower-level activities. This both helps reduce the impact of interleaving, and will prove useful for evaluation.

- An investigation into the use of modern techniques from the field of natural language processing (NLP) and related fields to discover activities in a novel manner, which allows for the explicit disentaglement of interleaved activities. The proposed approach indirectly models the interleaving observed in the data, allowing it to deal with interleaved data in a more principled manner than many existing systems can.

- An identification of a number of issues with existing “standard" approaches to evaluating activity discovery systems. To help rectify these issues, it is proposed to use a wider range of metrics from natural language processing (NLP) and the further incorporation of minimum description length (MDL) principles for AD evaluation.

What follows is an outline of how each of these contributions were covered in the thesis.

8.1.1 Interleaving

Most major datasets used for activity discovery and recognition, as well as most real-world data streams fed to activity discovery systems in a real-life application have the property of being interleaved (Stoia et al., 2008; Sagha et al., 2011; Singla et al., 2009). Interleaving is a difficult problem,

and one which existing activity discovery systems often struggle with, leading to often very complex systems being proposed to deal with it (Riboni et al., 2016). After the experience with the topic modelling system outlined in Chapter 3, it was decided that tackling the interleaving problem would be a major focus of this thesis. The proposed solution was built on the intuition that interleaving can be accounted for by machine learning models that are aware of long-range dependencies. Such models are already used in a range of tasks, most prominently for language modelling. The next step was to proceed to build such a system between Chapters 4 to Chapter 6. The system presented in Chapter 6 is the final iteration of this work, and shows impressive, state-of-the-art performance on interleaved datasets, as evidenced by the results presented in Section 6.3.

8.1.2 Hierarchical modelling of activities

The topic modelling system presented in Chapter 3 was the first activity discovery system built, and the first to be described in this thesis. Compared to the deep learning-based systems that were to follow, its performance was quite poor, but it still provided an important baseline. The system built on previous work from authors like Huynh et al. (2008), but with a number of simplifications. This demonstrated that it is possible build an activity discovery system that does not utilise temporal data directly, greatly simplifying the complexity of the proposed system.

This system also served as the first test of the idea of building a hierarchical structure, which would be a major part of later systems. Typically, authors make a distinction between *actions* (which are low-level behaviours like a single limb movement) and more abstract *activities* (sequences of actions that are carried out with the intention of achieving some goal). It could be argued that the situation is more complex than this, and that activities can themselves form a complex hierarchy in most cases. A large, complex activity will typically be composed of many smaller, simpler activities. This is an important realisation, because different activity discovery systems may produce outputs at differing levels of abstraction. Trying to find ways to output hierarchical structures is therefore important. This thesis has shown that there is a simple, general way to do this: replace events in the input dataset with discovered activities, which can be treated as new events, thus producing a new dataset that can be input again into the AD system in a hierarchical fashion. This process can continue as many times as needed to produce complex hierarchies of activities.

8.1.3 Language modelling and deep learning

Much recent progress in artificial intelligence and machine learning has been driven by deep learning. Deep learning hasn't had as big an impact on activity discovery however, which is unfortunate, since DL systems are often simpler to construct and use (in terms of end-to-end training

avoiding the need for hand-crafted feature design), and far more effective than more traditional machine learning models. In particular, much research has been done on producing models that can deal with long-term dependencies in sequential datasets. The system that was presented in Chapter 6 of this thesis shows just how useful such models can be.

Three models were presented in all: one which used a basic, feed-forward neural language model as proposed by (Bengio et al., 2003) in Chapter 4, a more sophisticated LSTM-based system in Chapter 5, and a final system incorporating LSTMs with a more realistic transitive link-building system in Chapter 6.

It is worth noting that the work presented in this thesis does not constitute a useful contribution to the field of NLP, since the nature of the datasets being used are quite different from natural language. However, the approach to activity discovery in this thesis is clearly an adaptation of existing NLP techniques. This shows that existing NLP models can be modified to allow them to model sequential datasets that are quite different to those a language model would generally be used with – in other words, neural language models are able to deal with a wide range of sequential datasets.

8.1.4 Evaluation

Existing evaluation metrics for activity discovery systems are often quite poor. Some of these problems were discussed, as well as possible solu-

tions to them, in Chapter 7. The chapter opened with an outline of existing techniques for activity discovery evaluation that have been documented in the literature. Some of these, such as compressive stability, seem quite promising and match the author's own intutions about evaluation.

Nonetheless, the evaluation of activity discovery systems remains an open problem, and it was concluded that activity discovery systems are best evaluated by a combination of metrics, rather than focusing on one as being authoritative. The chapter also proposed a new metric. One issue with the "standard" evaluation techniques is that the output of an activity discovery system may well be on a very different level of abstraction from the ground truth, or the output of another system. In Section 7.2.2, it was suggested that it might be possible to reduce the impact of this problem by extending the length of the activities in both the ground truth and the system output to match each other more closely. This thesis argued that this metric seems to be a fairer way to go about evaluating activity discovery systems (at least in cases where a ground truth exists for the dataset). In particular, it seems to deal with fragmented output better than other metrics, and can thus also be used to quantify the degree of fragmentation in the output produced by an activity discovery system.

8.2 Future work

There are a number of future research directions that could be explored from the work in this thesis. Some concrete proposals are:

8.2.1 Incorporate temporal information

All of the systems presented in this thesis have utilised simple binary sensor events. Many of the datasets that were used, however, are richer than this. In particular, most of them have *temporal information* – that is, they explicitly record *when* each sensor event occurred. Most activity discovery systems utilise this information as part of their discovery process. This thesis intentionally avoided using such information, preferring to work with the simpler binary datasets. However, modifying the systems to incorporate such information would likely be useful. For example, if two sensor events occur far apart in time, they are less likely to be related. Although good results have been achieved in this thesis without the use of temporal information, it is likely that incorporating that information would lead to futher performance improvements, since many activities are time-dependent (for example, a *Making-Dinner* activity should generally only occur in the evening, people may be out at work for much of the day and so on). Many other systems proposed in the literature (e.g. (Cook et al., 2013; Huynh et al., 2008)) deal with this problem using a sliding window that covers a period of time, rather than a fixed set of

events, for example.

8.2.2 Compare different network architectures

This is a somewhat obvious proposal, but that doesn't make it any less useful. Many different types of neural language model exist, and trying a number of them could be very beneficial. In particular, a number of different types of LSTM exist – LSTMs with and without a forget gate, peephole LSTMs, bidirectional LSTMs, gated recurrent units and so forth. Comparing these to one another could be useful. The LSTM used in Chapters 5 and 6 are known to work well with language, which is known to be similar to the domain discussed in this thesis (for example, the datasets used in the thesis tend to follow a Zipf-like probability distribution). However, the two are not identical – in particular, there is no real analogy to interleaving in language. Thus a comparison of network architectures, and an analysis of their errors, could be very fruitful.

8.2.3 Develop better clustering algorithms

The proposed method uses a clustering algorithm to lift activity instances into activity types. Improving this should lead to higher-quality activity types being found. The clustering algorithms used in this thesis are reasonable, but quite simplistic. It is possible that someone with more expertise in the field could improve upon them. This could be done based on temporal proximity, for instance, which would of course require

adding temporal information to the input dataset. Clustering rare event types into more common ones might also make learning higher layers easier. A more complex clustering method *was* tried in the past, but this turned out to perform poorly at discriminating between events from different activities.

8.2.4 Compute MDL tradeoffs

One major problem with deep learning systems is that the computational resources required are often very large. Arguably, this means that they should score poorly on a pure MDL-based metric. Recall that MDL ratings take two things into account: the degree to which the original input dataset was successfully learned and thus compressed, and the size of the final learned model. Since these models may be very large, this could mean that the final MDL value is very large, which is a bad thing. However, the maths haven't been done explicitly. Given a dataset $\mathbf{D}$ of length $L(\mathbf{D})$, used to train a model $\mathbf{M}$ of length $L(\mathbf{M})$, which compresses the dataset to a length of $L(\mathbf{D}|\mathbf{M})$, MDL practitioners would argue that $L(\mathbf{D}|\mathbf{M})$ must be less than $L(\mathbf{D})$ for it to be argued that the dataset has been learned correctly. Is this the case? If it isn't the case, but $\mathbf{M}$ exhibits a very high performance on other useful metrics like compressive stability, is that a acceptable tradeoff?

8.2.5 Other directions for future work

One interesting and unresolved question is why does the system presented in Chapter 6 perform *better* with interleaving? Intuitively, this seems to be a much harder problem. What in particular about this design causes that behaviour? In the domain of grammar induction, it is known that replacing words with their corresponding parts-of-speech can improve performance. Trying to find a way to convert an input dataset of events into part-of-speech-like categories (using techniques such as those presented by Eisner (2013)) could therefore have a similar positive effect on activity discovery. Finally, could real-world knowledge be incorporated into the model? If the system knows what real-world activity a sensor event type corresponds to, could it automatically label activities? For example, knowing a particular event type corresponds to turning the kettle on could allow for the automatic labelling of a *Making Tea* or *Making Coffee* activity. This information could potentially help with clustering, or perhaps by biasing the clustering process in favour of producing realistic activities. This information could possibly also be used in the evaluation phase, because it can give insights into the forms of the activities that are likely to occur in the dataset.

8.3 Final thoughts

Overall, it is hoped that this thesis will be a useful contribution to the activity discovery community, and will help push the state-of-the-art in the field. As noted in the introduction, activity discovery holds promise for automating the annotation of datasets for related fields such as activity recognition, and building models of human behaviour. The contributions of this thesis are intended to further these goals.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Angeli, G., Premkumar, M. J. J., and Manning, C. D. (2015). Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354.

Aztiria, A., Augusto, J. C., Basagoiti, R., Izaguirre, A., and Cook, D. J. (2012). Discovering frequent user–environment interactions in intelligent environments. *Personal and Ubiquitous Computing*, 16(1):91–103.

Bahl, L. R., Brown, P. F., de Souza, P. V., and Mercer, R. L. (1989). A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):1001–1008.

Bailey, T. L., Elkan, C., et al. (1994). Fitting a mixture model by expectation maximization to discover motifs in bipolymers. Technical report, University of California at San Diego.

Bakir, G., Hofmann, T., Schölkopf, B., Smola, A. J., and Taskar, B. (2007). *Predicting structured data*. MIT press.

Barron, A., Rissanen, J., and Yu, B. (1998). The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Blei, D. M., Griffiths, T. L., and Jordan, M. I. (2010). The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*, 57(2):1–30.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Buhler, J. and Tompa, M. (2002). Finding motifs using random projections. *Journal of computational biology*, 9(2):225–242.

Byrne, C. A., O'Grady, M., Collier, R., and O'Hare, G. M. (2020). An evaluation of graphical formats for the summary of activities of daily living (adls). In *Healthcare*, volume 8, page 194. Multidisciplinary Digital Publishing Institute.

Chen, W., Grangier, D., and Auli, M. (2015). Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*.

Chen, Y., Diethe, T., and Flach, P. (2016). Adltm: A topic model for discovery of activities of daily living in a smart home. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York, NY, USA*, pages 9–15.

Chiu, B., Keogh, E., and Lonardi, S. (2003). Probabilistic discovery of time series motifs. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–498.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124.

Cook, D. J. and Krishnan, N. C. (2015). *Activity learning: discovering, recognizing, and predicting human behavior from sensor data*. John Wiley & Sons.

Cook, D. J., Krishnan, N. C., and Rashidi, P. (2013). Activity discovery and activity recognition: A new partnership. *IEEE transactions on cybernetics*, 43(3):820–828.

Cook, D. J. and Schmitter-Edgecombe, M. (2009). Assessing the quality of activities in a smart environment. *Methods of information in medicine*, 48(05):480–485.

Déjean, H. (2000). Allis: a symbolic learning system for natural language learning. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 95–98. Association for Computational Linguistics.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22.

Dieng, A. B., Wang, C., Gao, J., and Paisley, J. (2016). Topicrnn: A recurrent neural network with long-range semantic dependency. *arXiv preprint arXiv:1611.01702*.

Dunn, J. C. (1974). Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104.

D'Ulizia, A., Ferri, F., and Grifoni, P. (2011). A survey of grammatical inference methods for natural language learning. *Artificial Intelligence Review*, 36(1):1–27.

Eisner, J. (2013). Deep learning of recursive structure: Grammar induction. Keynote talk at International Conference on Learning Representations.

Fox, I., Ang, L., Jaiswal, M., Pop-Busui, R., and Wiens, J. (2017). Contextual motifs: Increasing the utility of motifs using contextual data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 155–164.

Gjoreski, H. and Roggen, D. (2017). Unsupervised online activity discovery using temporal behaviour assumption. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pages 42–49.

Gold, E. M. (1967). Language identification in the limit. *Information and control*, 10(5):447–474.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Grunwald, P. (2004). A tutorial introduction to the minimum description length principle. *arXiv preprint math/0406077*.

Grünwald, P. D., Myung, I. J., and Pitt, M. A. (2005). *Advances in minimum description length: Theory and applications*. MIT press.

Hammerla, N. Y., Fisher, J., Andras, P., Rochester, L., Walker, R., and Plötz, T. (2015). Pd disease state assessment in naturalistic environments using deep learning. In *Twenty-Ninth AAAI conference on artificial intelligence*.

Hammid, R., Maddi, S., Johnson, A., Bobick, A., Essa, I., and Isbell, C. L. (2012). Unsupervised activity discovery and characterization from event-streams. *arXiv preprint arXiv:1207.1381*.

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Horn, D., Solan, Z., Ruppin, E., and Edelman, S. (2004). Unsupervised language acquisition: syntax from plain corpus. In *Presented at the Newcastle Workshop on Human Language*.

Huynh, T., Fritz, M., and Schiele, B. (2008). Discovery of activity patterns using topic models. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 10–19.

Jelinek, F., Mercer, R. L., Bahl, L. R., and Baker, J. K. (1977). Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63.

Jenks, G. F. (1967). The data model concept in statistical mapping. *International yearbook of cartography*, 7:186–190.

Kamijo, K.-i. and Tanigawa, T. (1990). Stock price pattern recognition-a recurrent neural network approach. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 215–221. IEEE.

Katz, S., Downs, T. D., Cash, H. R., and Grotz, R. C. (1970). Progress in development of the index of adl. *The gerontologist*, 10(1_Part_1):20–30.

Kelleher, J. D. (2019). *Deep Learning*. MIT Press ISBN: 9780262537551.

Khedekar, D. C., Truco, A. C., Oteyza, D. A., and Huertas, G. F. (2017). Home automation—a fast-expanding market. *Thunderbird International Business Review*, 59(1):79–91.

Kim, E., Helal, S., and Cook, D. (2009). Human activity recognition and pattern discovery. *IEEE pervasive computing*, 9(1):48–53.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *Thirtieth AAAI conference on artificial intelligence*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kwapisz, J. R., Weiss, G. M., and Moore, S. A. (2011). Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82.

Lane, N. D., Georgiev, P., and Qendro, L. (2015). Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 283–294.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Li, H. and Yuan, B. (1998). Chinese word segmentation. In *Proceedings of the 12th Pacific Asia Conference on Language, Information and Computation*, pages 212–217.

Li, M., O'Grady, M., Gu, X., Alawlaqi, M. A., O'Hare, G., et al. (2018). Time-bounded activity recognition for ambient assisted living. *IEEE transactions on emerging topics in computing*.

Liu, D. and Yu, J. (2009). Otsu method and k-means. In *2009 Ninth International Conference on Hybrid Intelligent Systems*, volume 1, pages 344–349. IEEE.

Liu, S., Zhao, Y., Xue, F., Chen, B., and Chen, X. (2019). Deepcount: Crowd counting with wifi via deep learning. *arXiv preprint arXiv:1903.05316*.

Lonardi, J. and Patel, P. (2002). Finding motifs in time series. In *Proc. of the 2nd Workshop on Temporal Data Mining*, pages 53–68.

Machanick, P. and Bailey, T. L. (2011). Meme-chip: motif analysis of large dna datasets. *Bioinformatics*, 27(12):1696–1697.

Mahalunkar, A. and Kelleher, J. D. (2018a). Understanding recurrent neural architectures by analyzing and synthesizing long distance dependencies in benchmark sequential datasets. *arXiv preprint arXiv:1810.02966*.

Mahalunkar, A. and Kelleher, J. D. (2018b). Using regular languages to explore the representational capacity of recurrent neural architectures. In *International Conference on Artificial Neural Networks*, pages 189–198. Springer.

Mahalunkar, A. and Kelleher, J. D. (2019). Multi-element long distance dependencies: Using spk languages to explore the characteristics of long-distance dependencies. *arXiv preprint arXiv:1907.06048*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239. IEEE.

Mnih, A. and Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*.

Mohan, A., Choksi, M., and Zaveri, M. A. (2019). Anomaly and activity recognition using machine learning approach for video based surveillance. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6. IEEE.

Nevill-Manning, C. G. and Witten, I. H. (1997). Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82.

Nguyen, T., Zhang, Q., Le, D. V., and Karunanithi, M. (2017). Dirichlet process gaussian mixture model for activity discovery in smart homes with ambient sensors. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 383–392.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66.

Petasis, G., Paliouras, G., Karkaletsis, V., Halatsis, C., and Spyropoulos, C. D. (2004). e-grids: Computationally efficient gramatical inference from positive examples. *Grammars*, 7:69–110.

Pevzner, P. A., Sze, S.-H., et al. (2000). Combinatorial approaches to finding subtle signals in dna sequences. In *ISMB*, volume 8, pages 269–278.

Plank, B., Søgaard, A., and Goldberg, Y. (2016). Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *arXiv preprint arXiv:1604.05529*.

Raeiszadeh, M., Tahayori, H., and Visconti, A. (2019). Discovering varying patterns of normal and interleaved adls in smart homes. *Applied Intelligence*, 49(12):4175–4188.

Rajasekaran, S., Balla, S., and Huang, C.-H. (2005). Exact algorithms for planted motif problems. *Journal of Computational Biology*, 12(8):1117–1128.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing*.

Ravi, N., Dandekar, N., Mysore, P., and Littman, M. L. (2005). Activity recognition from accelerometer data. In *Aaai*, volume 5, pages 1541–1546.

Riboni, D., Sztyler, T., Civitarese, G., and Stuckenschmidt, H. (2016). Unsupervised recognition of interleaved activities of daily living through ontological and probabilistic reasoning. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 1–12.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–471.

Rogers, E., Kelleher, J. D., and Ross, R. J. (2016). Using topic modelling algorithms for hierarchical activity discovery. In *Using topic modelling algorithms for hierarchical activity discovery*, pages 41–48. Springer.

Rogers, E., Kelleher, J. D., and Ross, R. J. (Forthcoming (due for publication in June 2020)a). Language model co-occurrence linking for interleaved activity discovery. In *Machine Learning for Networking*.

Rogers, E., Ross, R. J., and Kelleher, J. D. (2018). Evaluating sequence discovery systems in an abstraction-aware manner. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 261–272. Springer.

Rogers, E., Ross, R. J., and Kelleher, J. D. (2020b). Modelling interleaved activities using language models. In *ECMS*, pages 183–189.

Roggen, D., Calatroni, A., Rossi, M., Holleczek, T., Förster, K., Tröster, G., Lukowicz, P., Bannach, D., Pirkl, G., Ferscha, A., et al. (2010). Collecting complex activity datasets in highly rich networked sensor environments. In *2010 Seventh international conference on networked sensing systems (INSS)*, pages 233–240. IEEE.

Ross, R. and Kelleher, J. (2013). A comparative study of the effect of sensor noise on activity recognition models. In *International Joint Conference on Ambient Intelligence*, pages 151–162. Springer.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Ruotsalainen, M., Ala-Kleemola, T., and Visa, A. (2007). Gais: A method for detecting interleaved sequential patterns from imperfect data. In *2007 IEEE Symposium on Computational Intelligence and Data Mining*, pages 530–534. IEEE.

Safavi, T., Fourney, A., Sim, R., Juraszek, M., Williams, S., Friend, N., Koutra, D., and Bennett, P. N. (2020). Toward activity discovery in the personal web. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 492–500.

Sagha, H., Digumarti, S. T., Millán, J. d. R., Chavarriaga, R., Calatroni, A., Roggen, D., and Tröster, G. (2011). Benchmarking classification techniques using the opportunity human activity dataset. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 36–40. IEEE.

Salton, G., Ross, R., and Kelleher, J. (2017). Attentive language models. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 441–450.

Seiter, J., Amft, O., Rossi, M., and Tröster, G. (2014). Discovery of activity composites using topic models: An analysis of unsupervised methods. *Pervasive and Mobile Computing*, 15:215–227.

Singla, G., Cook, D. J., and Schmitter-Edgecombe, M. (2009). Tracking activities in complex settings using smart environment technologies. *International journal of biosciences, psychiatry, and technology (IJBSPT)*, 1(1):25.

Sipser, M. (2012). *Introduction to the Theory of Computation (3rd edition)*. Cengage Learning. ISBN: 978-1133187790.

Smith, N. A. (2011). Linguistic structure prediction. *Synthesis lectures on human language technologies*, 4(2):1–274.

Solan, Z., Horn, D., Ruppin, E., and Edelman, S. (2005). Unsupervised learning of natural languages. *Proceedings of the National Academy of Sciences*, 102(33):11629–11634.

Sporns, O. and Kötter, R. (2004). Motifs in brain networks. *PLoS biology*, 2(11).

Stoia, L., Shockley, D. M., Byron, D. K., and Fosler-Lussier, E. (2008). Scare: a situated corpus with annotated referring expressions. In *LREC*. Citeseer.

Sun, L., Zhang, D., Li, B., Guo, B., and Li, S. (2010). Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations. In *International conference on ubiquitous intelligence and computing*, pages 548–562. Springer.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Tenorth, M., Bandouch, J., and Beetz, M. (2009). The tum kitchen data set of everyday manipulation activities for motion tracking and action recognition. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1089–1096. IEEE.

Thapa, K., Al, A., Md, Z., Lamichhane, B., and Yang, S.-H. (2020). A deep machine learning method for concurrent and interleaved human activity recognition. *Sensors*, 20(20):5770.

Vahdatpour, A., Amini, N., and Sarrafzadeh, M. (2009). Toward unsupervised activity discovery using multi dimensional motif detection in time series. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Van Kasteren, T., Noulas, A., Englebienne, G., and Kröse, B. (2008). Accurate activity recognition in a home setting. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 1–9.

Van Zaanen, M. (2000). Abl: Alignment-based learning. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*, pages 961–967. Association for Computational Linguistics.

Viard, K., Fanti, M. P., Faraut, G., and Lesage, J.-J. (2020). Human activity discovery and recognition using probabilistic finite-state automata. *IEEE Transactions on Automation Science and Engineering*.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269.

Wang, J., Chen, Y., Hao, S., Peng, X., and Hu, L. (2019). Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11.

Wang, W., Seraj, F., and Havinga, P. J. (2020). A sound-based crowd activity recognition with neural network based regression models. In *Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, pages 1–8.

Ward, J. A., Lukowicz, P., and Tröster, G. (2006). Evaluating performance in continuous context recognition using event-driven error characterisation. In *International Symposium on Location-and Context-Awareness*, pages 239–255. Springer.

Weiss, G. M., Timko, J. L., Gallagher, C. M., Yoneda, K., and Schreiber, A. J. (2016). Smartwatch-based activity recognition: A machine learning approach. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 426–429. IEEE.

Welch, P. B. (2013). *Chinese art: A guide to motifs and visual imagery*. Tuttle Publishing.

Wu, J., Wang, L., Wang, L., Guo, J., and Wu, G. (2019). Learning actor relation graphs for group activity recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9964–9974.

Xu, H., Pan, Y., Li, J., Nie, L., and Xu, X. (2019). Activity recognition method for home-based elderly care service based on random forest and activity similarity. *IEEE Access*, 7:16217–16225.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zhao, Z., Koutsopoulos, H. N., and Zhao, J. (2020). Discovering latent activity patterns from transit smart card data: A spatiotemporal topic model. *Transportation Research Part C: Emerging Technologies*, 116:102627.